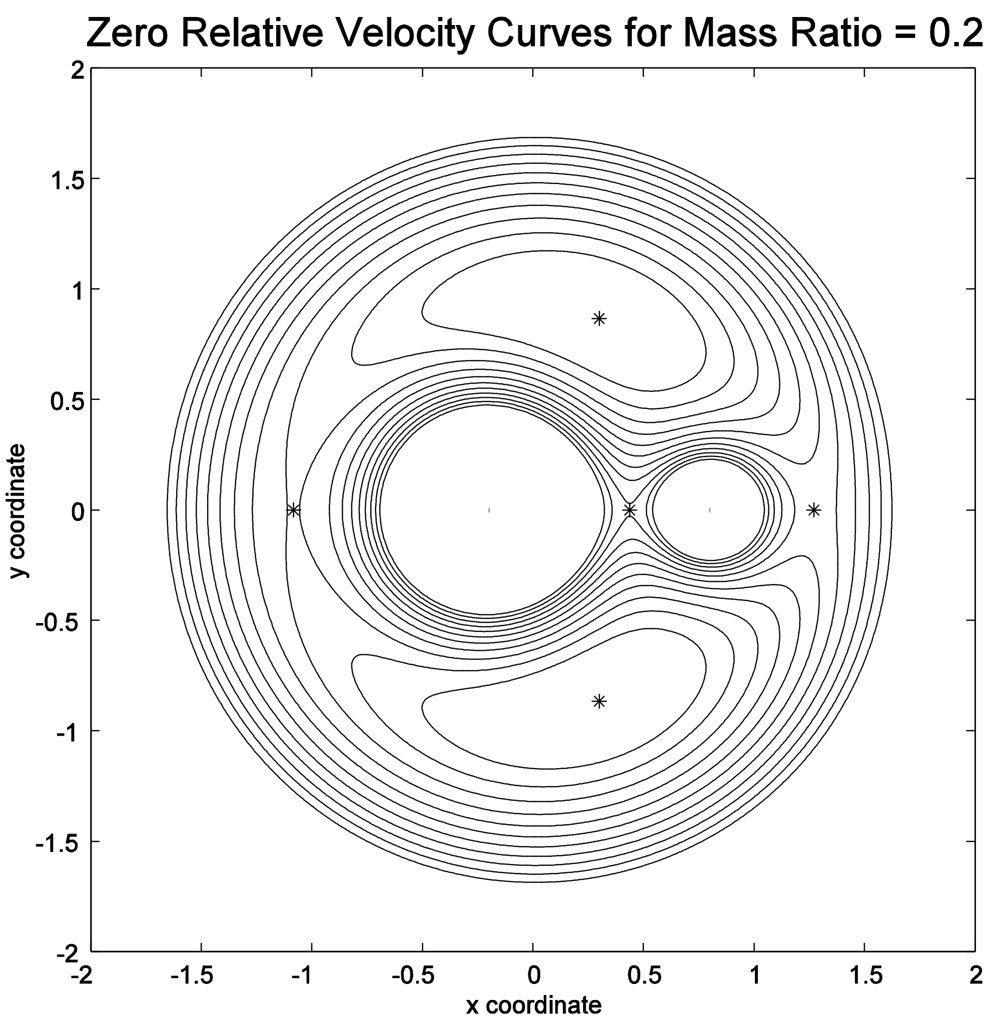
Celestial Computing with MATLAB



Introduction

Celestial Computing with MATLAB is a comprehensive collection of MATLAB functions, scripts and data files that can be used to solve a variety of problems in fundamental and advanced celestial mechanics. This suite of applications and functions can be used to solve practical problems in the following areas of computational celestial mechanics:

- Rise and Set of the Sun, Moon and Planets
- Lunar and Solar Eclipses
- Lunar Occultations
- Transits of Mercury and Venus
- Equinoxes and Solstices
- The Circular-Restricted Three-Body Problem
- Cowell’s and Encke’s Method for Heliocentric Objects
- Apparent Coordinates of the Sun, Moon, Planets and Stars
- Kepler’s Equation
- Two-body Orbital Motion

This MATLAB toolbox also contains many numerical methods, basic celestial mechanics functions and data files that can be used to create your own applications. The following is a summary of these routines:

- Date and time routines
- Lunar, solar and planetary ephemerides
- Astronomical Coordinates and Transformations
- Root-finding and optimization
- Differential equations of heliocentric orbital motion

The *Celestial Computing with MATLAB* software suite also includes binary data files for the SLP96 and DE403 ephemeris. Please note that the binary ephemeris and map files provided with this software package are only compatible with the Windows version of MATLAB.

Requirements

Celestial Computing with MATLAB requires MATLAB version 5.3 or higher.

1. Rise and Set of the Sun, Moon and Planets

This MATLAB script (`riseset.m`) determines rise and set conditions of the Sun, Moon and planets. This software uses a combination of one-dimensional minimization and root-finding to predict visibility. The source ephemeris for this routine is the JPL DE405.

The basic procedure for predicting rise and set of a celestial body involves the following computational steps and MATLAB functions:

(1) locate an *extrema* using the functions `oevent1` and `minima`

(2) bracket a "forward" and "backward" root using the function `broot`

(3) find each root using the function `brent`

(4) calculate the event circumstances using the function `events1`

(5) display the event circumstances using the function `rsfunc`

The main MATLAB function that solves this problem has the following syntax and arguments:

```
function oevent1 (objfunc, prtfunc, ti, tf, dt, dtsml)

% predict minimization/root-finding events

% input

%  objfunc = objective function
%  prtfunc = display results function
%  ti      = initial simulation time
%  tf      = final simulation time
%  dt      = step size used for bounding minima
%  dtsml   = small step size used to determine whether
%            the function is increasing or decreasing
```

The name of the user-defined objective function (`objfunc`) and a function that prints the important results (`prtfunc`) is passed to this routine in the argument list.

Choosing the Search Parameters

The proper selection of the search parameters `dt` and `dtsml` for this algorithm depends on the type of celestial event you are trying to predict. For example, the value of `dt` is used to bound one or more *extremas* and depends on how long the event lasts. For shorter events `dt` should be smaller and for longer events `dt` can be larger. Be careful not to make `dt` too large or the algorithm may "step over" one or more solutions. For lunar events `dt` can be between 0.1 and 0.25 days and for planetary events `dt` can be between 0.1 and 0.5 days.

The value of `dtsml` must be selected such that it produces a "big" enough change in the objective function to tell the algorithm if it is moving "downhill" or "uphill". This is similar to choosing the value of *x* when numerically estimating the derivative of a function of the form $y = f(x)$. For planetary events a value of `dtsml = 0.1` days should be adequate. For lunar events this number should be a number between `0.01` and `0.05` days. An examination of a plot of the objective function over a period of time can also provide insight into the proper selection of the `dtsml` parameter.

During the search process the `oevent1` function calls a MATLAB function called `minima` that calculates the extremum of a one-dimensional user-defined objective function. The syntax of this optimization function is as follows:

```
function [xmin, fmin] = minima (f, a, b, tolm)

% one-dimensional minimization

% Brent's method

% input

%  f    = objective function coded as y = f(x)
%  a    = initial x search value
%  b    = final x search value
%  tolm = convergence criterion

% output

%  xmin = minimum x value
%  fmin = minimum function value
```

This function in turn requires an objective function coded as `fx = objfunc(x)` where `objfunc` is the name of the user-defined objective function, `x` is the current function argument and `fx` is the objective function evaluated at `x`. For this script the objective function is equal to the topocentric elevation angle of the body. A numerical value of `1e-4` for the *convergence criterion* should be adequate for most user-defined objective functions.

The source code of the objective function for this example is as follows:

```
function fx = rsfunc(x)

% topocentric elevation angle
% objective function

% input

%  x = elpased simulation time (days)

% output

%  fx = objective function at x

% Celestial Computing with MATLAB

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global jdatei itarg icent rootflg

global obslat glon glat ht stlocl
global aunit elevation azimuth

rtd = 180 / pi;

dtr = pi / 180;

jdutc = jdatei + x;

jdtdt = utc2tdt(jdutc);

[ra, dec, dis] = applan1(jdtdt, jdutc, itarg, icent, 1, glon, glat, ht);

% apparent hour angle

hangle = stlocl - (2 * pi * ra / 24);

% topocentric elevation (degrees)

elevation = rtd * asin(sin(obslat) * sin(dtr * dec) ...
            + cos(obslat) * cos(dtr * dec) * cos(hangle));

% topocentric azimuth (degrees)

tmp1 = sin(hangle);

tmp2 = cos(hangle) * sin(obslat) - tan(dtr * dec) * cos(obslat);

azimuth = mod(180 + rtd * atan3(tmp1, tmp2), 360);

if (rootflg == 1)
   % correct for horizontal refraction

   dref = (34 / 60);

   if (itarg == 11)
      % correct for semidiameter of the moon

      tsdia = rtd * asin((1738 / aunit) / dis);
   elseif (itarg == 10)
      % correct for semidiameter of the sun

      tsdia = rtd * asin((0.5 * 696000 / aunit) / dis);
   else
      % no semidiameter correction for planets
      tsdia = 0;
   end
else
   dref = 0;

   tsdia = 0;
end

% evaluate objective function

fx = -(elevation + tsdia + dref);
```

Notice the use of a *root flag* `rootflg` to “toggle” the function when finding a minimum (`rootflg = 0`) or a root (`rootflg = 1`) of the objective function.

This function calculates the topocentric elevation angle of the celestial object for any input argument `x`. For this script the input argument is the elapsed time in days since the simulation began. Notice the corrections for the semidiameter of the Moon and the horizontal refraction during the root-finding calculations.

The syntax of the MATLAB function that calculates and displays the actual roots of the user-defined objective function is as follows:

```
function events1 (objfunc, prtfunc, ti, tf, topt)

% compute and display celestial events

% input

%  objfunc = objective function
%  prtfunc = display results function
%  ti      = initial simulation time
%  tf      = final simulation time
%  topt    = extrema time
```

This function is a "driver" routine that uses the time of the extrema `topt` to first bracket the backward and forward roots and then actually find the value of each root. After each root is calculated this routine then calls a MATLAB function called `rsfunc` that actually displays the event conditions.

The following is a brief description of the function that brackets each root.

function broot

This MATLAB function is used to bracket a single root of a single nonlinear equation of the form $y = f(x)$ It uses a combination of *geometric acceleration* and *time rectification* to bracket single roots. The basic idea is to find the endpoints of a time interval x_1 and x_2 such that $f(x_1) f(x_2) < 0$. This condition guarantees that there is at least one root in the interval because the objective function changes sign.

The user must supply this function with an initial guess for x_1 and x_2. Typically x_1 is the time of an objective function minimum or maximum and x_2 is a time `0.05` days after (forward root) or `0.05` days before (backward root) the value of x_1. The sign of this delta tells the algorithm in which direction to move.

The name of the user-defined objective function `f` is passed to this routine in the argument list. For the `riseset` script the user-defined (and coded) objective function is called `rsfunc`.

The syntax of this MATLAB function is as follows:

```
function [x1out, x2out] = broot (f, x1in, x2in, factor, dxmax)

% bracket a single root of a nonlinear equation

% input

%  f      = objective function coded as y = f(x)
%  x1in   = initial guess for first bracketing x value
%  x2in   = initial guess for second bracketing x value
%  factor = acceleration factor (non-dimensional)
%  dxmax  = rectification interval

% output

%  x1out = final value for first bracketing x value
%  x2out = final value for second bracketing x value
```

The acceleration factor used in this script is `0.25` and the rectification interval is `0.1` days. The acceleration factor increases the size of each step geometrically and the rectification interval monitors the length of the current bracketing interval. When necessary the rectification logic reinitializes the search and prevents the interval from becoming too large and skipping one or both ends of the bracketing interval completely. The size of each step increases geometrically until the length of the current step reaches the value of the rectification interval. At that point the step size is reset and the process begins again. Eventually, the process will bracket the root and the function will return the endpoints of this bracketing interval.

The following is a brief description of the MATLAB function that solves for each root.

function brent

This function uses Brent's method to find a single root of a single nonlinear equation of the form $y = f(x)$. Derivatives are not required for this algorithm. However, the user should ensure that a root is bracketed, $f(x_1) f(x_2) < 0$, before calling this routine. Additional information about this numerical method can be found in *Algorithms for Minimization Without Derivatives*, R. P. Brent, Prentice-Hall, 1972. As the title of this book indicates, this algorithm does not require derivatives of the objective function. This feature is important because the analytic first derivative of many objective functions is difficult to derive and code.

The syntax of this MATLAB function is as follows:

```
function [xroot, froot] = brent (f, x1, x2, rtol)

% solve for a single real root of a nonlinear equation

% Brent's method

% input

%  f    = objective function coded as y = f(x)
%  x1   = lower bound of search interval
%  x2   = upper bound of search interval
%  rtol = algorithm convergence criterion

% output

%  xroot  = real root of f(x) = 0
%  froot  = function value at f(x) = 0
```

The root-finding performance of this algorithm is determined by `rtol`. A value of `1.0e-8` should be adequate for most problems. Smaller values will predict roots more accurately at the expense of longer program execution times.

The `riseset` MATLAB script will prompt you for an initial calendar date, on the UTC time scale, and the search interval in days. It will also ask you for the geographic coordinates of the observer. Please note that north latitudes are positive and south latitudes are negative. Also note that east longitudes are positive and west longitudes are negative. Furthermore, observer sites above sea level have positive altitudes and sites below sea level are negative.

The following is a typical user interaction with this script. It illustrates two rise and sets of the Moon. The initial calendar date was January 1, 1998, and the observer was located at 40° north latitude and 105° west longitude. The altitude of the site is 0 meters. The calendar date and time displayed is on the UTC time scale.

```
rise and set of the sun, moon or planets


please input an initial UTC calendar date

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


please input the search duration (days)
? 5

    target body menu

  <1> Mercury
  <2> Venus
  <3> Earth
  <4> Mars
  <5> Jupiter
  <6> Saturn
  <7> Uranus
  <8> Neptune
  <9> Pluto
  <10> Sun
  <11> Moon

please select the target body
? 11

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 40,0,0

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -105,0,0

please input the altitude of the ground site (meters)
```

```
(positive above sea level, negative below sea level)
? 0

rise and set conditions

    'Moon'


observer latitude        +40d 00m 00.00s

observer east longitude  -105d 00m 00.00s

observer altitude              0.0000 meters


rise conditions

calendar date        01-Jan-1998

universal time       16:27:02

UTC Julian date      2450815.1854

azimuth angle        +107d 16m 34.84s

elevation angle      +00d -50m 08.02s

   < please press any key to continue >


maximum elevation conditions

calendar date        01-Jan-1998

universal time       21:55:47

UTC Julian date      2450815.4137

azimuth angle        +180d 44m 55.48s

elevation angle      +36d 55m 51.41s

   < please press any key to continue >


 set conditions

calendar date        02-Jan-1998

universal time       03:25:20

UTC Julian date      2450815.6426

azimuth angle        +254d 49m 56.04s

elevation angle      +00d -50m 09.37s

event duration       +10h 58m 17.6827s

   < please press any key to continue >


rise conditions
```

```
   calendar date        02-Jan-1998

   universal time       17:07:18

   UTC Julian date      2450816.2134

   azimuth angle        +102d 17m 18.07s

   elevation angle      +00d -50m 10.40s

      < please press any key to continue >

   maximum elevation conditions

   calendar date        02-Jan-1998

   universal time       22:49:50

   UTC Julian date      2450816.4513

   azimuth angle        +180d 53m 01.40s

   elevation angle      +40d 55m 36.02s

      < please press any key to continue >

   set conditions

   calendar date        03-Jan-1998

   universal time       04:33:46

   UTC Julian date      2450816.6901

   azimuth angle        +260d 17m 02.22s

   elevation angle      +00d -50m 10.77s

   event duration       +11h 26m 27.6213s
```

This same technique can be used to predict rise and set of stars and other stellar objects. For this case simply call the `apstar1` or `apstar2` MATLAB functions from within the `rsfunc` objective function.

2. Lunar Eclipses

This MATLAB script (`eclipse.m`) can be used to predict lunar eclipses. The source ephemeris for this script is SLP96. However, the code can be easily modified to use DE405 as the source ephemeris. This software provides the eclipse type, the universal times and topocentric coordinates of the Moon at the beginning and end of the penumbra contacts, and the time and coordinates at maximum eclipse.

This script uses a combination of one-dimensional minimization and root-finding to solve this classic problem. The objective function used in these calculations involves the geocentric separation angle between the center of the Moon and the anti-Sun position vector or shadow axis, and the semidiameter and horizontal parallax of the Sun and Moon. This function is given by the following expression:

$$f\left(t\right) = \cos^{-1}\left(-\hat{\mathbf{U}}_m \bullet \hat{\mathbf{U}}_s\right) - f_1 + s_m$$

where

$\hat{\mathbf{U}}_s =$ geocentric unit position vector of the Sun

$\hat{\mathbf{U}}_m =$ geocentric unit position vector of the Moon

$f_1 = 1.02\left(\pi_1 + \pi_s + s_s\right) =$ size of penumbra shadow

$\pi_1 = 0.99834\pi_m =$ corrected parallax

$\pi_m =$ horizontal parallax of the Moon

$\pi_s =$ horizontal parallax of the Sun

$s_m =$ semidiameter of the Moon

$s_s =$ semidiameter of the Sun

If we let σ represent the minimum geocentric separation angle of the Moon relative to the shadow axis, a penumbral lunar eclipse occurs whenever the following geometric condition is satisfied:

$$\sigma < 1.02\left(\pi_1 + \pi_s + s_s\right) + s_m$$

A partial lunar eclipse will happen whenever the following is true:

$$\sigma < 1.02\left(\pi_1 + \pi_s - s_s\right) + s_m$$

The geometric condition for a total lunar eclipse is given by

$$\sigma < 1.02\left(\pi_1 + \pi_s - s_s\right) - s_m$$

In these expressions

$$\pi_m = \sin^{-1}\left(\frac{r_{eq}}{d_m}\right) \qquad \pi_s = \sin^{-1}\left(\frac{r_{eq}}{d_s}\right)$$

$$s_m = \sin^{-1}\left(\frac{r_m}{d_m}\right) \qquad s_s = \sin^{-1}\left(\frac{r_s}{d_s}\right)$$

where r_{eq} is the equatorial radius of the Earth (6378.14 kilometers), r_m is the radius of the Moon (1738 kilometers), r_s is the radius of the Sun (696,000 kilometers), d_m is the geocentric distance of the Moon and d_s is the geocentric distance of the Sun.

The following is a typical user interaction with this script. It illustrates the circumstances of the total lunar eclipse of November 28, 1993. The topocentric coordinates are for an observer located in Denver, Colorado.

```
              lunar eclipses

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 11,1,1993

please input the search duration (days)
? 30

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 40,0,0

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -105,0,0

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 0

working ...


 total lunar eclipse

 begin penumbral phase of lunar eclipse

calendar date             29-Nov-1993

universal time            03:27:08

UTC Julian date           2449320.6438

lunar azimuth angle        +99d 45m 53.83s

lunar elevation angle      +43d 53m 09.29s

   < please press any key to continue >
working ...
```

```
 greatest eclipse conditions

calendar date             29-Nov-1993

universal time            06:26:13

UTC Julian date           2449320.7682

lunar azimuth angle        +164d 18m 16.73s

lunar elevation angle      +70d 13m 39.01s

   < please press any key to continue >


working ...

 end penumbral phase of lunar eclipse

calendar date             29-Nov-1993

universal time            09:25:00

UTC Julian date           2449320.8924


lunar azimuth angle        +250d 53m 05.61s

lunar elevation angle      +52d 24m 42.53s

event duration             +05h 57m 51.8379s
```

3. Lunar Occultations

This MATLAB script (`occult.m`) can be used to predict the local circumstances of lunar occultations of a planet or star. The source ephemeris for this script is SLP96. However, the code can be easily modified to use DE405 as the source ephemeris.

This script uses a combination of one-dimensional minimization and root-finding to solve this problem. The objective function used in these calculations is the topocentric separation angle between the center of the Moon and the object being occulted. This function is given by the following expression:

$$f(t) = \cos^{-1}\left(\hat{\mathbf{U}}_m \bullet \hat{\mathbf{U}}_b\right) - \left(s_m + s_b\right)$$

where

$\hat{\mathbf{U}}_m =$ topocentric unit position vector of the Moon

$\hat{\mathbf{U}}_b =$ topocentric unit position vector of the object

$s_m =$ semidiameter of the Moon

$s_b =$ semidiameter of the object

The semidiameter of the Moon is given by

$$s_m = \sin^{-1}\left(\frac{r_m}{d_m}\right)$$

where r_m is the radius of the Moon (1738 kilometers) and d_m is the topocentric distance of the Moon. The semidiameter of a star is zero and the semidiameter of a planet is

$$s_p = \frac{s_{p_0}}{r_p}$$

where s_{p_0} is the semidiameter of the planet at a distance of 1 astronomical unit and r_p is the topocentric distance of the planet.

The following is a typical user interaction with this script. It illustrates a lunar occultation of the star Spica relative to an observer located in Denver, Colorado.

```
                lunar occultations

   please input the calendar date
   (1 <= month <= 12, 1 <= day <= 31, year = all digits!)
   ? 2,1,1987


   please input the search duration (days)
```

```
? 30

    target body menu

  <1> Mercury
  <2> Venus
  <3> Mars
  <4> Jupiter
  <5> Saturn
  <6> Uranus
  <7> Neptune
  <8> Pluto
  <9> Star

please select the target body
? 9

please input the star data file name
(be sure to include the file name extension)
? spica.dat

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 39,45,0

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -104,58,48

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 0

working ...

 begin lunar occultation

calendar date             18-Feb-1987

universal time            12:27:04

UTC Julian date           2446845.0188


lunar azimuth angle        +214d 02m 41.91s

lunar elevation angle      +32d 52m 22.10s

   < please press any key to continue >

working ...

 end lunar occultation

calendar date             18-Feb-1987

universal time            13:17:46

UTC Julian date           2446845.0540
```

```
lunar azimuth angle        +226d 06m 05.32s

lunar elevation angle      +26d 31m 56.13s

event duration             +00h 50m 41.6772s

   < please press any key to continue >
```

The following is the star data file used in this example.

```
star name
SPICA

J2000 right ascension (hours)
13.4198852780

J2000 declination (degrees)
-11.1613083330

J2000 proper motion in right ascension (seconds/Julian century)
-0.2780

J2000 proper motion in declination (arcseconds/Julian century)
-2.8300

parallax (arcseconds)
0.0210

radial velocity (kilometers/second)
1.0
```

4. Solar Eclipses

This MATLAB script (`seclipse.m`) can be used to predict local circumstances of solar eclipses. The source ephemeris for this script is SLP96. However, the code can be easily modified to use DE405 as the source ephemeris. This software provides the universal times and topocentric coordinates of the Moon at the beginning and end of the penumbra contacts, and the time and coordinates at maximum eclipse.

This script uses a combination of one-dimensional minimization and root-finding to solve this problem. The objective function used in these calculations is the selenocentric separation angle between the axis of the shadow cast by the Moon and an Earth observer. This function is given by the following expression:

$$f(t) = \cos^{-1}\left(\hat{\mathbf{U}}_{axis} \bullet \hat{\mathbf{U}}_{m-o}\right) - \psi_p$$

where

$\hat{\mathbf{U}}_{axis} =$ selenocentric unit vector of the Moon's shadow

$\hat{\mathbf{U}}_{m-o} =$ selenocentric unit position vector of the observer

$\psi_p =$ penumbra shadow angle

The penumbra shadow angle at the distance of the Earth observer is determined from

$$\psi_p = \sin^{-1}\left(\frac{r_m}{d_m}\right) + \sin^{-1}\left(\frac{r_s + r_m}{d_{m-s}}\right)$$

In this expression r_m is the radius of the Moon, r_s is the radius of the Sun, d_m is the topocentric distance of the Moon and d_{m-s} is the distance from the Moon to the Sun.

The selenocentric position vector of the Sun is computed from the expression

$$\mathbf{R}_{m-s} = \mathbf{R}_s - \mathbf{R}_m$$

where $\mathbf{R}_s$ is the geocentric position vector of the Sun and $\mathbf{R}_m$ is the geocentric position vector of the Moon.

The following is a typical user interaction with this script. This example is taken from page 209 of *Astronomy with the Personal Computer*.

```
                solar eclipses

   please input the calendar date
   (1 <= month <= 12, 1 <= day <= 31, year = all digits!)
   ? 5,1,1994

   please input the search duration (days)
```

```
? 30

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 33,57,0

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -6,50,0

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 0

working ...

 begin penumbral phase of solar eclipse

calendar date              10-May-1994

universal time             17:50:46

UTC Julian date            2449483.2436

lunar azimuth angle         +280d 49m 06.97s

lunar elevation angle       +15d 58m 45.39s

   < please press any key to continue >

working ...

 greatest eclipse conditions

calendar date              10-May-1994

universal time             18:58:15

UTC Julian date            2449483.2904

lunar azimuth angle         +289d 25m 24.94s

lunar elevation angle       +02d 55m 27.28s

   < please press any key to continue >

working ...

 end penumbral phase of solar eclipse

calendar date              10-May-1994

universal time             19:59:20

UTC Julian date            2449483.3329

lunar azimuth angle         +297d 56m 19.07s

lunar elevation angle       -08d 14m 55.69s

event duration              +02h 08m 34.2403s
```

5. Transits of Mercury and Venus

This MATLAB script (`transit.m`) can be used to predict the local circumstances of transits of Mercury and Venus. The source ephemeris for this script is SLP96. However, the code can be easily modified to use DE405 as the source ephemeris.

This script uses a combination of one-dimensional minimization and root-finding to solve this problem. The objective function used in these calculations is the topocentric separation angle between the center of the Sun and the planet in transit. This function is given by the following expression:

$$f\left(t\right) = \cos^{-1}\left(\hat{\mathbf{U}}_s \bullet \hat{\mathbf{U}}_p\right) - \left(s_s + s_p\right)$$

where

$\hat{\mathbf{U}}_s =$ topocentric unit position vector of the Sun

$\hat{\mathbf{U}}_p =$ topocentric unit position vector of the planet

$s_s =$ semidiameter of the Sun

$s_p =$ semidiameter of the planet

During the search for minimum separation angles, the software also checks to see if the planet is between the observer and the Sun.

The semidiameter of the Sun is given by

$$s_s = \sin^{-1}\left(\frac{r_s}{d_s}\right)$$

where r_s is the radius of the Sun (696,000 kilometers) and d_s is the topocentric distance of the Sun. The semidiameter of Mercury and Venus is determined from the expression

$$s_p = \frac{s_{p_0}}{r_p}$$

where s_{p_0} is the semidiameter of the planet at a distance of 1 astronomical unit and r_p is the topocentric distance of the planet. This script uses a value of 3.36 arc seconds for Mercury and 8.41 arc seconds for Venus.

The following is a typical user interaction with this script. This example is Example 6 from the book *Transits* by Jean Meeus. The software will also display the topocentric coordinates of the Sun at the time of ingress, least separation distance and egress. The ingress and egress conditions correspond to exterior contact.

```
transits of Mercury and Venus

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 6,1,2012

please input the search duration (days)
? 30

 target body menu

  <1> Mercury

  <2> Venus

please select the target body
? 2

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 38,55,17

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -77,3,56

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 84

working ...

 ingress - exterior contact - Venus

calendar date             05-Jun-2012

universal time            22:03:45

UTC Julian date           2456084.4193


azimuth angle of the sun        +279d 05m 05.27s

elevation angle of the sun      +25d 56m 12.81s

   < please press any key to continue >

working ...

 least distance of the Sun and Venus

calendar date             06-Jun-2012

universal time            01:26:18

UTC Julian date           2456084.5599

azimuth angle of the sun        +309d 48m 31.41s

elevation angle of the sun      -09d 40m 16.84s
```

```
   < please press any key to continue >

working ...

 egress - exterior contact - Venus

calendar date             06-Jun-2012

universal time            04:51:00

UTC Julian date           2456084.7021

azimuth angle of the sun        +355d 50m 23.84s

elevation angle of the sun      -28d 16m 44.34s

event duration                  +06h 47m 15.4074s
```

6. Closest Approach Between the Earth and Heliocentric Objects

This MATLAB script (`cae2ho.m`) uses a combination of Cowell's method and one-dimensional minimization to predict closest approach conditions between the Earth and objects such as asteroids, comets and other bodies in heliocentric elliptic orbits. It includes the point mass gravity perturbations due to the planets Mercury, Venus, Earth, Mars, Jupiter, Saturn and the Sun. It also includes the point-mass effect of the Moon via a combined Earth/Moon gravitational constant.

The user can elect to use either the JPL DE405 ephemeris or the SLP96 ephemeris developed by the Bureau of Longitudes in Paris for the planetary and solar ephemeris calculations. The software expects to find one or both of these binary files in the same directory as `cae2ho`. Additional information about the SLP96 ephemeris can be found on the Internet at **ftp://ftp.bdl.fr/pub/ephem/sun/slp96**. Information about the DE405 ephemeris is available at **ftp://navigator.jpl.nasa.gov**.

The second-order heliocentric equations of motion of a satellite or celestial body subject to the point mass gravitational attraction of the Sun and planets are given by

$$\ddot{\mathbf{r}} = \frac{d^2\mathbf{r}}{dt^2}\left(\mathbf{r},t\right) = -\mu_s \frac{\mathbf{r}_{s-b}}{\left|\mathbf{r}_{s-b}\right|^3} - \sum_{i=1}^{9} \mu_{p_i} \left(\frac{\mathbf{r}_{(p-b)_i}}{\left|\mathbf{r}_{(p-b)_i}\right|^3} + \frac{\mathbf{r}_{p_i}}{\left|\mathbf{r}_{p_i}\right|^3} \right)$$

where

μ_s = gravitational constant of the Sun

μ_{p_i} = gravitational constant of planet *i*

$\mathbf{r}_p$ = position vector from the Sun to planet

$\mathbf{r}_{s-b}$ = position vector from the Sun to the body

$\mathbf{r}_{p-b}$ = position vector from the planet to the body

These position vectors are related according to

$$\mathbf{r}_{s-b} = \mathbf{r}_p + \mathbf{r}_{p-b}$$

This program uses the Runge-Kutta-Fehlberg 7(8) method to numerically integrate the first-order form of the orbital equations of motion. This is a variable step size method of order 7 with an 8th order error estimate that is used to dynamically change the integration step size during the simulation.

This software also uses a one-dimensional optimization algorithm due to Richard Brent to solve the close approach problem. Additional information about this numerical method can be found in *Algorithms for Minimization Without Derivatives*, R.P. Brent, Prentice-Hall, 1972. As the title of this book indicates, this algorithm does not require derivatives of the *objective function*. This feature is important because the analytic first derivative of many objective functions is difficult to derive and code. The objective function for this program is the scalar geocentric distance of the heliocentric object.

The software will ask you for the name of a simple ASCII input file that defines the simulation. When responding to this request be sure to include the file name extension. The following is a typical input data file for this program. It contains the *J2000 equator and equinox* heliocentric orbital elements of the asteroid 1997 XF 11. This file was created using data available on the Horizons ephemeris system which is located on the Internet at **http://ssd.jpl.nasa.gov/horizons.html**. Do not change the total number of lines or the order of annotation and data in this file. The software expects to find exactly 31 lines of information in the input data file.

```
initial calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
10,20,1998

initial TDB
(0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
0,0,0

initial semimajor axis (AU)
(semimajor axis > 0)
0.1441779254846407D+01

initial orbital eccentricity (non-dimensional)
(0 <= eccentricity < 1)
0.4836740409964595D+00

initial orbital inclination (J2000 equator and equinox)
(degrees; 0 <= inclination <= 180)
0.2017287833484568D+02

initial argument of perihelion (J2000 equator and equinox)
(degrees; 0 <= argument of perihelion <= 360)
0.3228014264370198D+03

initial right ascension of the ascending node (J2000 equator and equinox)
(degrees; 0 <= raan <= 360)
0.3533298230482567D+03

initial mean anomaly (J2000 equator and equinox)
(degrees; 0 <= mean anomaly <= 360)
0.2708752703925974D+03
```

Please note the proper units and coordinate system (J2000 equinox and equator) of the input orbital elements. Be sure to include all the digits of the calendar year. Notice also that time on the TDB scale is used.

The following is a typical user interaction with this MATLAB script. The user can specify the search duration in days and a minimum close approach distance in Astronomical Units.

```
                       program cae2ho

< closest approach between the Earth and a heliocentric object >
```

```
      < please press any key to continue >

   please input the name of the data file
   (be sure to include the file name extension)
   ? xf11.dat

   please input the search duration (days)
   ? 2000

   would you like to enforce a minimum distance constraint (y = yes, n = no)
   ? y

   please input the minimum distance constraint (AU)
   ? .1

   please select the ephemeris source (1 = DE405, 2 = SLP96)
   ? 1
```

The software will display the initial conditions as follows:

```
   initial heliocentric orbital elements
   (J2000/ICRF equinox and equator)

         sma (au)        eccentricity     inclination (deg)    argper (deg)
    1.4417792548e+000  4.8367404100e-001  2.0172878335e+001  3.2280142644e+002

        raan (deg)     true anomaly (deg)   arglat (deg)       period (days)
    3.5332982305e+002  2.2165440865e+002  1.8445583509e+002  7.3186814210e-003

      < please press any key to continue >


   DE405 ephemeris

   time and conditions at closest approach

   calendar date        31-Oct-2002
   TDB time             00:26:38

   TDB Julian date      2452578.5185

   geocentric distance      0.06350013  AU

   geocentric distance    9499483.83935515  km


   heliocentric orbital elements
   (J2000/ICRF equinox and equator)

         sma (au)        eccentricity     inclination (deg)    argper (deg)
    1.4422695673e+000  4.8412378833e-001  2.0165272969e+001  3.2279202361e+002

        raan (deg)     true anomaly (deg)   arglat (deg)       period (days)
    3.5334764382e+002  7.7110770918e+001  3.9902794530e+001  7.3224150847e-003

      < please press any key to continue >
```

For this example the output is with respect to the J2000/ICRF equinox and equator. When using SLP96 as the source ephemeris, the output will be with respect to the J2000 dynamical equinox and equator.

7. Equinoxes and Solstices

This MATLAB script (`equsol.m`) determines the time of the equinoxes and solstices of the Earth. These events are the times when the apparent geocentric longitude of the Sun is an exact multiple of 90 degrees. This script uses Brent's root-finder and the `sun1.m` solar ephemeris to calculate these events.

Brent's method requires an objective function that defines the nonlinear equation to be solved. The objective function for the spring and fall equinoxes is the geocentric declination of the Sun. For the summer and winter solstices the objective function is $\theta - \lambda_s$, where $\theta = 90^\circ$ for the summer solstice, $\theta = 270^\circ$ for the winter solstice and λ_s is the geocentric apparent longitude of the Sun. Brent's method also requires an initial and final time which bounds the root of the objective function. The initial time for the spring equinox is March 15, for the summer solstice June 15, for the fall equinox September 15 and for the winter solstice December 15. For each event, the final time is equal to these initial dates plus 15 days.

The `equsol` script will prompt you for the calendar date of interest. The following is a typical user interaction with this script. Please note that the event is displayed in dynamical or ephemeris time.

```
time of the equinoxes and solstices


please input the calendar year
? 2000


   time of the equinoxes and solstices

spring equinox

calendar date         20-Mar-2000
dynamical time          07:36:11

summer solstice

calendar date         21-Jun-2000
dynamical time          01:48:48

fall equinox

calendar date         22-Sep-2000
dynamical time          17:28:45

winter solstice

calendar date         21-Dec-2000
dynamical time          13:38:30
```

8. Cowell's Method for Heliocentric Orbits

cowell1.m – rectangular position and velocity vector formulation

This MATLAB script (`cowell1.m`) uses Cowell's method to propagate the motion of spacecraft, comets and other celestial bodies in heliocentric elliptic orbits. It includes the point mass gravity perturbations due to the planets Venus, Earth, Mars, Jupiter, Saturn and the Sun. The low precision ephemerides used in this application can be replaced with the DE405 or SLP96 algorithms.

The second-order heliocentric equations of motion of a satellite or celestial body subject to the point mass gravitational attraction of the Sun and planets are given by

$$\ddot{\mathbf{r}} = \frac{d^2\mathbf{r}}{dt^2}(\mathbf{r},t) = -\mu_s \frac{\mathbf{r}_{s-b}}{\left|\mathbf{r}_{s-b}\right|^3} - \sum_{i=1}^{9} \mu_{p_i} \left(\frac{\mathbf{r}_{(p-b)_i}}{\left|\mathbf{r}_{(p-b)_i}\right|^3} + \frac{\mathbf{r}_{p_i}}{\left|\mathbf{r}_{p_i}\right|^3} \right)$$

where

$\mu_s =$	gravitational constant of the Sun
$\mu_{p_i} =$	gravitational constant of planet *i*
$\mathbf{r}_p =$	position vector from the Sun to planet
$\mathbf{r}_{s-b} =$	position vector from the Sun to the body
$\mathbf{r}_{p-b} =$	position vector from the planet to the body

These position vectors are related according to

$$\mathbf{r}_{s-b} = \mathbf{r}_p + \mathbf{r}_{p-b}$$

The following data file defines a simulation for Halley's comet.

```
calendar date of perihelion passage
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
2,9,1986

universal time of perihelion passage
(0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
15,52,14.592

perihelion radius (Astronomical Units)
(perihelion radius > 0)
0.587478

orbital eccentricity (non-dimensional)
(0 <= eccentricity < 1)
0.967329

orbital inclination (degrees)
(0 <= inclination <= 180)
```

```
162.2486

argument of perigee (degrees)
(0 <= argument of perigee <= 360)
111.8114

right ascension of the ascending node (degrees)
(0 <= raan <= 360)
58.1291
```

The syntax of the MATLAB function that reads this file is as follows:

```
function [fid, oev1, jdpp] = readoe2(filename)

% read orbital elements data file

% required by encke.m and cowell2.m

% input

%  filename = name of orbital element data file

% output

%  fid    = file id
%  jdpp   = julian date of perihelion passage
%  oev(1) = semimajor axis
%  oev(2) = orbital eccentricity
%  oev(3) = orbital inclination
%  oev(4) = argument of perigee
%  oev(5) = right ascension of the ascending node
%  oev(6) = true anomaly
```

The following is a typical user interaction with this script.

```
program cowell1

< heliocentric orbit propagation using Cowell's method >

   < please press any key to continue >

 simulation data menu

   <1> user input

   <2> data file

? 2


please input the simulation data file name
(be sure to include the file name extension)
? halley.dat

initial calendar date and universal time

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,1986

please input the universal time
```

```
(0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
? 0,0,0

please input the simulation period (days)
? 120

please input the algorithm error tolerance
(a value of 1.0e-10 is recommended)
? 1e-8
```

The following is the output created for this example:

```
                    program cowell1

    < heliocentric orbital motion - Cowell's method >


final conditions

calendar date      01-May-1986

universal time     00:00:00

      sma (km)         eccentricity     inclination (deg)    argper (deg)
 2.6903161323e+009  9.6733201594e-001  1.6224898626e+002  1.1181332479e+002

     raan (deg)     true anomaly (deg)   arglat (deg)       period (days)
 5.8130641784e+001  1.0750909426e+002  2.1932241905e+002  2.7855777571e+004


perihelion radius   5.8748967570e-001 AU
```

cowell2.m – modified equinoctial orbital elements formulation

This MATLAB script (`cowell2.m`) determines the long-term orbit evolution of objects in heliocentric orbits using Cowell's solution of the modified equinoctial orbital elements form of the differential equations of motion. The user can provide the initial orbital elements for this program interactively or by specifying the name of a data file as described in the cowell1.m script above.

The following is a typical user interaction with this script.

```
                  program cowell2

< heliocentric orbit propagation using Cowell's method >


 simulation data menu

   <1> user input

   <2> data file

? 2


please input the simulation data file name
(be sure to include the file name extension)
```

```
? halley.dat

initial calendar date and universal time

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,1986

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

please input the simulation period (days)
? 120

please input the algorithm error tolerance
(a value of 1.0e-8 is recommended)
? 1e-8
```

The following is the data display for this example.

```
                      program cowell2

    < heliocentric orbital motion - Cowell's method >


final conditions

calendar date      01-May-1986

universal time     00:00:00

      sma (km)        eccentricity     inclination (deg)    argper (deg)
 2.6903158654e+009  9.6733201271e-001  1.6224898624e+002  1.1181332505e+002

     raan (deg)     true anomaly (deg)   arglat (deg)       period (days)
 5.8130641942e+001  1.0750909422e+002  2.1932241926e+002  2.7855773424e+004


perihelion radius   5.8748967534e-001 AU
```

Technical Discussion

The relationship between modified equinoctial orbital elements 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 equations of orbital motion expressed in terms of modified equinoctial orbital elements are as follows:

$$\dot{p} = \frac{dp}{dt} = \frac{2p}{w}\sqrt{\frac{p}{\mu}}\,\Delta_t$$

$$\dot{f} = \frac{df}{dt} = \sqrt{\frac{p}{\mu}}\left[\Delta_r \sin L + \left[\left(w+1\right)\cos L + f\right]\frac{\Delta_t}{w} - \left(h\sin L - k\cos L\right)\frac{g\Delta_n}{w}\right]$$

$$\dot{g} = \frac{dg}{dt} = \sqrt{\frac{p}{\mu}}\left[-\Delta_r \cos L + \left[\left(w+1\right)\sin L + g\right]\frac{\Delta_t}{w} + \left(h\sin L - k\cos L\right)\frac{g\Delta_n}{w}\right]$$

$$\dot{h} = \frac{dh}{dt} = \sqrt{\frac{p}{\mu}}\,\frac{s^2\Delta_n}{2w}\cos L$$

$$\dot{k} = \frac{dk}{dt} = \sqrt{\frac{p}{\mu}}\,\frac{s^2\Delta_n}{2w}\sin L$$

$$\dot{L} = \frac{dL}{dt} = \sqrt{\mu p}\left(\frac{w}{p}\right)^2 + \frac{1}{w}\sqrt{\frac{p}{\mu}}\left(h\sin L - k\cos L\right)\Delta_n$$

where $\Delta_r, \Delta_t, \Delta_n$ are external perturbations in the radial, tangential and orbit normal directions, respectively.

The equations of motion can also be expressed as follows:

$$\dot{\mathbf{y}} = \frac{d\mathbf{y}}{dt} = \mathbf{A}\left(\mathbf{y}\right)\mathbf{P} + \mathbf{b}$$

where

$$\mathbf{A}(\mathbf{y}) = \begin{Bmatrix} 0 & \dfrac{2p}{q}\sqrt{\dfrac{p}{\mu}} & 0 \\ \sqrt{\dfrac{p}{\mu}}\sin L & \sqrt{\dfrac{p}{\mu}}\dfrac{1}{q}\left\{(q+1)\cos L + f\right\} & -\sqrt{\dfrac{p}{\mu}}\dfrac{g}{q}\left\{h\sin L - k\cos L\right\} \\ -\sqrt{\dfrac{p}{\mu}}\cos L & \sqrt{\dfrac{p}{\mu}}\left\{(q+1)\sin L + g\right\} & \sqrt{\dfrac{p}{\mu}}\dfrac{f}{q}\left\{h\sin L - k\cos L\right\} \\ 0 & 0 & \sqrt{\dfrac{p}{\mu}}\dfrac{s^2\cos L}{2q} \\ 0 & 0 & \sqrt{\dfrac{p}{\mu}}\dfrac{s^2\sin L}{2q} \\ 0 & 0 & \sqrt{\dfrac{p}{\mu}}\left\{h\sin L - k\cos L\right\} \end{Bmatrix}$$

and

$$\mathbf{b} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \sqrt{\mu p}\left(\dfrac{q}{p}\right)^2 \end{Bmatrix}$$

In this equation **P** is the total disturbing acceleration vector given by

$$\mathbf{P} = \Delta_r \hat{\mathbf{i}}_r + \Delta_t \hat{\mathbf{i}}_t + \Delta_n \hat{\mathbf{i}}_n$$

In this MATLAB script these perturbations consist of the point-mass gravity effects of the planets. Other perturbations such as solar radiation pressure can also be included.

Furthermore, $\hat{\mathbf{i}}_r, \hat{\mathbf{i}}_t$ and $\hat{\mathbf{i}}_n$ are unit vectors in the radial, tangential and normal directions given by

$$\hat{\mathbf{i}}_r = \frac{\mathbf{r}}{\|\mathbf{r}\|}$$

$$\hat{\mathbf{i}}_n = \frac{\mathbf{r} \times \mathbf{v}}{\|\mathbf{r} \times \mathbf{v}\|}$$

$$\hat{\mathbf{i}}_t = \hat{\mathbf{i}}_n \times \hat{\mathbf{i}}_r$$

9. Encke’s Method for Heliocentric Orbits

This MATLAB application (`encke.m`) demonstrates heliocentric orbit propagation using Encke’s method. It includes the point mass gravity perturbations due to the planets Venus, Earth, Mars, Jupiter, Saturn and the Sun.

The classic form of Encke’s special perturbation method solves the initial value problem (IVP) of celestial mechanics by analytically propagating a two-body “reference” orbit and numerically integrating the deviations from two-body motion. These two procedures are performed for the same time interval or step size. Since the two-body solution is “exact”, the time interval used is usually a function of the numerical integration technique and the magnitude of the perturbations. After each propagation interval, the errors between the reference orbit and the perturbed orbit are evaluated. When these errors become “large”, a process known as *rectification of the orbit* is performed. This involves adding the integrated deviations to the two-body orbit to produce a new reference orbit called the osculating orbit. After rectification, the algorithm is reinitialized with the elements of the new osculating orbit and the process is repeated until either the errors become large once more or the final time is reached.

The numerical algorithm used in this script simplifies this technique by using a variable step-size integration method to automatically select the propagation step size. Since this type of integrator “senses” the magnitude of the perturbations, it will choose the “best” and largest step size for each propagation interval. The user can control how well this procedure is performed with the algorithm truncation error tolerance. Furthermore, the software eliminates the logic required to determine when rectification should occur by simply rectifying the orbit after each and every propagation step.

The two-body or unperturbed equations of motion of a spacecraft or celestial body are

$$\frac{d^2\mathbf{r}_k}{dt^2} + \mu \frac{\mathbf{r}_k(t)}{r_k^3} = 0$$

where $\mathbf{r}_k$ is the unperturbed position vector of the spacecraft at any time t and μ is the gravitational constant of the primary body.

The equations of motion of motion of a spacecraft subject to a point mass central body and other external perturbations are

$$\frac{d^2\mathbf{r}_p}{dt^2} + \mu \frac{\mathbf{r}_p(t)}{r_p^3} = \mathbf{a}(\mathbf{r}_p, t)$$

where $\mathbf{r}_p$ is the perturbed position vector at any time t and $\mathbf{a}(\mathbf{r}_p, t)$ is the vector of perturbations or other accelerations.

The difference between these two types of orbital motion is

$$\frac{\delta^2 \mathbf{r}}{dt^2} = \frac{d^2 \mathbf{r}_p}{dt^2} - \frac{d^2 \mathbf{r}_k}{dt^2}$$

In order to avoid numerical difficulties when subtracting small differences of vector equations, the following form of the equations of motion is used:

$$\frac{\delta^2 \mathbf{r}}{dt^2} = -\frac{\mu}{\rho^3}\left[f(q)\mathbf{r}_p(t) + \mathbf{r}_p(t) \right] + \mathbf{a}\left(\mathbf{r}_p, t\right)$$

where

$$f(q) = q\left(\frac{3 + 3q + q^2}{1 + (1+q)^{3/2}} \right)$$

and

$$q = \frac{(\delta\mathbf{r} - 2\mathbf{r}) \bullet \delta\mathbf{r}}{r^2}$$

At any time the position and velocity vectors of the true orbit are given by the vector sum of the two body and perturbed components as follows:

$$\mathbf{r}(t) = \mathbf{r}_k(t) + \mathbf{r}_p(t)$$

$$\mathbf{v}(t) = \mathbf{v}_k(t) + \mathbf{v}_p(t)$$

The gravitational acceleration on the spacecraft due to each planet is determined from the following expression:

$$\mathbf{a}(\mathbf{r}, t) = -\mu\left(\frac{\mathbf{r}_{p-b}}{\left|\mathbf{r}_{p-b}\right|^3} + \frac{\mathbf{r}_p}{\left|\mathbf{r}_p\right|^3} \right)$$

where

$\mathbf{r}_{p-b}$ = position vector of spacecraft relative to planet

$\mathbf{r}_{\mathrm{p}}$ = position vector of spacecraft relative to Sun

μ = gravitational constant of planet

The software uses a slightly modified version of Shepperd's two body algorithm to provide the unperturbed solution for any simulation time. This MATLAB file is called `twobody4` and has for its input and output the entire state vector in a single array instead of individual position and velocity arrays. The “deviation” equations of motion are integrated using a slightly modified version of the Runge-Kutta-Fehlberg 7(8) method called `rkencke`.

Although simple and straightforward, this approach does require special attention to the initialization at the start of each propagation step. Since there are no perturbations at the initial epoch of the osculating reference orbit, the integrator would use the initial step size defined by the user. This will cause very large errors before rectification can be performed. To avoid this problem, the software forces the algorithm to perform the first phase of the orbit propagation with a fixed step size. This is accomplished by setting the initial step size guess to a small value and the truncation error tolerance to a very large value. For the second phase of the propagation, the step size estimate is increased and the RKF algorithm is allowed to change the actual step size to the largest value based on the perturbations and user-defined error tolerance. The truncation error tolerance for this phase is set to a very small number. The "best" initial step size depends on the type of orbit propagation and the initial conditions.

Additional information about this algorithm can be found in AAS 92-196, "A Simple and Efficient Computer Implementation of Encke's Method", AAS/AIAA Spaceflight Mechanics Meeting, Colorado Springs, Colorado, February 24-26, 1992.

The information required for this script can either be manually input by the user or the software can read a simple data file of orbital elements. For both options the program will always ask the user for the initial calendar date and Universal Time of the simulation, the simulation duration in days, and the algorithm error tolerance. A tolerance of `1.0e-8` is recommended.

The following is a typical user interaction with this program.

```
                 program encke

  < orbit propagation using Encke's method >

     < please press any key to continue >

   simulation data menu

     <1> user input

     <2> data file

  ? 2


  please input the simulation data file name
  (be sure to include the file name extension)
  ? halley.dat

  initial calendar date and universal time

  please input the calendar date
  (1 <= month <= 12, 1 <= day <= 31, year = all digits!)
  ? 1,1,1986

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

```
please input the simulation period (days)
? 120

please input the algorithm error tolerance
(a value of 1.0e-8 is recommended)
? 1e-8
```

This program also uses the `readoe2` function to process the input data file. The following is the screen display created by this software for this example.

```
final conditions

calendar date      01-May-1986

universal time     00:00:00

      sma (km)        eccentricity     inclination (deg)    argper (deg)
 2.6903155747e+009  9.6733200916e-001  1.6224898624e+002  1.1181332495e+002

     raan (deg)     true anomaly (deg)   arglat (deg)       period (days)
 5.8130641692e+001  1.0750909431e+002  2.1932241926e+002  2.7855768910e+004


perihelion radius   5.8748967589e-001 AU
```

The low precision ephemeris used in this application can be replaced with the DE405 or SLP96 algorithms very easily.

10. The Circular-Restricted Three-Body Problem

This section describes several MATLAB scripts that can be used to analyze and display orbits in the circular-restricted three-body problem (CRTBP). This is a special case of the general three-body problem where the primary and secondary bodies move in circular orbits about the common center of mass, and the effect of the gravitational attraction of the smallest body and any other perturbations such as solar radiation pressure are ignored.

The applications described here are for orbital problems in the Earth-Moon-satellite system. However, the equations and algorithms are valid for any circular-restricted three-body system for which the mass ratio of the primary and secondary is between 0 and 1/2.

The classic textbook for this problem is *Theory of Orbits* by Victor Szebehely, Academic Press, New York, 1967. An excellent discussion about this problem can also be found in Chapter 8 of *An Introduction to Celestial Mechanics* by Forest Ray Moulton, Dover Publications, 1970.

Technical Discussion

The motion of the spacecraft or celestial body is modeled in a coordinate system which is rotating about the center-of-mass or barycenter of the Earth-Moon system. The motion of all three bodies is confined to the *x-y* plane. In the discussion that follows, subscript 1 is the Earth and subscript 2 is the Moon.

For convenience, the problem is formulated in *canonical* or non-dimensional units. The unit of length is taken to be the constant distance between the Earth and Moon, and the unit of time is chosen such that the Earth and Moon have an angular velocity ω about the barycenter equal to 1.

Kepler's third law for this formulation is given by

$$\omega^2 \left| m_1 m_2 \right|^3 = g\left(m_1 + m_2\right) = 1$$

The *x* and *y* coordinates of the Earth and Moon in this coordinate system are given by

$$x_1 = -\mu \qquad y_1 = 0$$

$$x_2 = 1 - \mu \qquad y_2 = 0$$

The position and velocity vectors of the small body in this system are sometimes called "synodical" coordinates to distinguish them from "sidereal" coordinates defined in an inertial, non-rotating coordinate system.

The system of second-order *non-dimensional* differential equations of motion of a point-mass satellite or celestial body in this rotating coordinate system are given by

$$\frac{d^2x}{dt^2} - 2\frac{dy}{dt} = x - (1-\mu)\frac{x - x_1}{r_1^3} - \mu\frac{x - x_2}{r_2^3}$$

$$\frac{d^2y}{dt^2} + 2\frac{dx}{dt} = y - (1-\mu)\frac{y}{r_1^3} - \mu\frac{y}{r_2^3}$$

$$\frac{d^2z}{dt^2} = -(1-\mu)\frac{z}{r_1^3} - \mu\frac{z}{r_2^3}$$

where

$x = x$ component of position

$y = y$ component of position

$z = z$ component of position

$x_1 = -\mu$

$x_2 = 1 - \mu$

$\mu =$ Earth-Moon mass ratio $= m_1 / m_2 \approx 1/81.27$

$r_1^2 = (x - x_1)^2 + y^2 + z^2$

$r_2^2 = (x - x_2)^2 + y^2 + z^2$

The *Jacobi integral* for the three-body problem is given by

$$V^2 = x^2 + y^2 + \frac{2(1-\mu)}{r_1} + \frac{2\mu}{r_2} - C$$

In this equation *V* is the scalar speed of the small body and *C* is a constant of integration called the *Jacobi constant*.

crtbp1.m – coordinates and energy of the libration points

In his prize memoir of 1772, Joseph-Louis Lagrange proved that there are five *equilibrium points* in the circular-restricted three-body problem. If a satellite or celestial body is located at one of these points with zero initial velocity relative to the rotating coordinate system, it will stay there permanently provided there are no perturbations. These special positions are also called *libration* points or stationary Lagrange points. Three of these libration points are on the *x*-axis (collinear points) and the other two form equilateral triangles with the positions of the Earth and Moon.

This MATLAB script can be used to calculate the coordinates and energy of the five libration or equilibrium points of the CRTBP. For the collinear libration points this script solves three nonlinear equations.

The nonlinear equation for the L_1 libration point is given by

$$x-\frac{1-\mu}{(\mu+x)^2}+\frac{\mu}{(x-1+\mu)^2}=0$$

The nonlinear equation for the L_2 libration point is

$$x-\frac{1-\mu}{(\mu+x)^2}-\frac{\mu}{(x-1+\mu)^2}=0$$

and the nonlinear equation for the L_3 collinear libration point is

$$x+\frac{1-\mu}{(\mu+x)^2}+\frac{\mu}{(x-1+\mu)^2}=0$$

The lower and upper bounds used during the root-finding search for all three equations are $x=-2$ and $x=+2$ respectively.

The L_4 libration point is located at $x=1/2-\mu,\ y=\sqrt{3}/2$ and the L_5 libration point is at $x=1/2-\mu,\ y=-\sqrt{3}/2$. For these two libration points $r_1=r_2=1$.

The following is a typical user interaction with this script.

```
              program crtbp1

   < equilibrium coordinates and energy >


   please input the value for the mass ratio
   ? .01
```

The following is the program output for this example.

```
                            program crtbp1

               < equilibrium coordinates and energy >


 mass ratio = 1.0000000000e-002

 location      x-coordinate       y-coordinate            energy

    L1           0.848079           0.000000       -1.5838206546e+000
    L2           1.146765           0.000000       -1.5771597543e+000

    L3          -1.004167           0.000000       -1.5049988584e+000

    L4           0.490000           0.866025       -1.4950500000e+000

    L5           0.490000          -0.866025       -1.4950500000e+000
```

g3body.m – graphics display of three-body motion

This MATLAB script graphically displays motion of a spacecraft or small celestial body in the circular-restricted three-body problem. This algorithm and program examples are based on the methods described in “Periodic Orbits in the Restricted Three-Body Problem with Earth-Moon Masses”, by R. A. Broucke, JPL TR 32-1168, 1968.

This script begins by displaying the following main menu.

```
               program g3body

   < graphics display of three-body motion >


          main menu

    <1> periodic orbit about L1

    <2> periodic orbit about L2

    <3> periodic orbit about L3

    <4> user input of initial conditions

    selection (1, 2, 3 or 4)
```

The first three menu options will display typical *periodic* orbits about the respective libration point. The initial conditions for each of these example orbits are as follows:

(1) periodic orbit about the L_1 libration point

$$\begin{aligned} x_0 &= 0.300000161 \\ \dot{y}_0 &= -2.536145497 \\ \mu &= 0.012155092 \\ t_f &= 5.349501906 \end{aligned}$$

(2) periodic orbit about the L_2 libration point

$$x_0 = 2.840829343$$
$$\dot{y}_0 = -2.747640074$$
$$\mu = 0.012155085$$
$$t_f = 11.933318588$$

(3) periodic orbit about the L_3 libration point

$$x_0 = -1/600000312$$
$$\dot{y}_0 = 2.066174572$$
$$\mu = 0.012155092$$
$$t_f = 6.303856312$$

Notice that each orbit is defined by an initial *x*-component of position, x_0, an initial *y*-component of velocity, $\dot{y}_0$, a value of Earth-Moon mass ratio μ, and an orbital period t_f. The initial *y*-component of position and *x*-component of velocity for each of these orbits is equal to zero. The software will draw the location of the Earth, Moon and libration point on the graphics screen.

If you would like to experiment with your own initial conditions, select option `<4>` from the main menu. The program will then prompt you for the initial *x* and *y* components of the orbit's position and velocity vector, and the values of μ to use in the simulation. The software will also ask for an initial and final time to run the simulation, and an integration step size. The value `0.01` is a good number for step size.

Here is a short list of initial conditions for several other periodic orbits that you may want to display with this script.

(1) Retrograde periodic orbit about m_1

$$x_0 = -2.499999883$$
$$\dot{y}_0 = 2.100046263$$
$$\mu = 0.012155092$$
$$t_f = 11.99941766$$

(2) Direct periodic orbit about m_1

$$x_0 = 0.952281734$$
$$\dot{y}_0 = -0.957747254$$
$$\mu = 0.012155092$$
$$t_f = 6.450768946$$

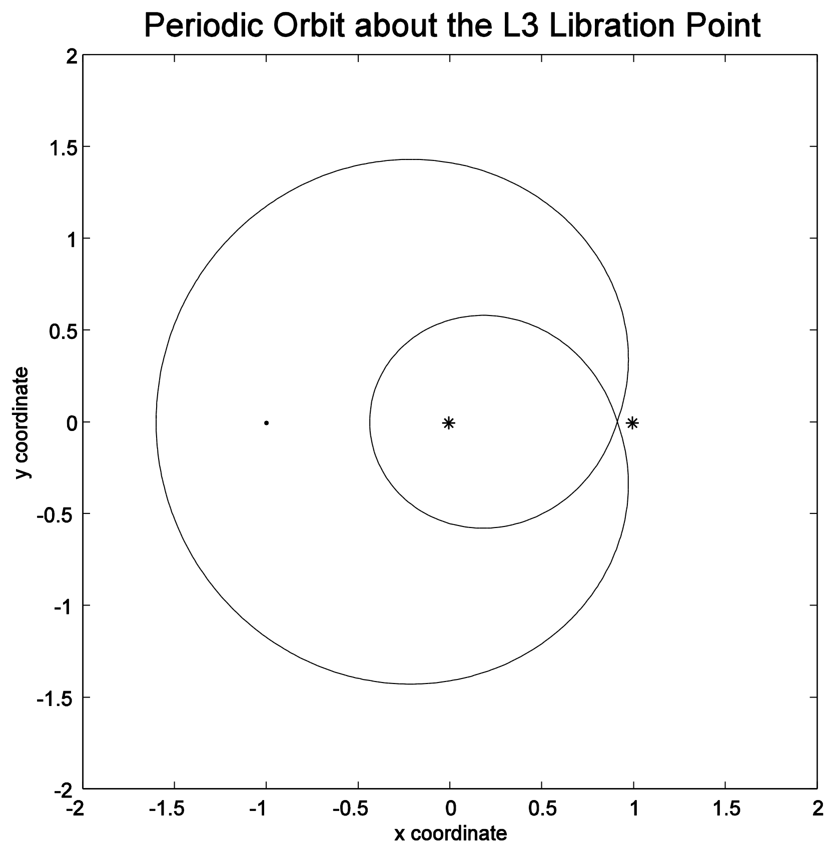
(3) Direct periodic orbit about m_1 and m_2

$$x_0 = 3.147603117$$
$$\dot{y}_0 = -3.07676285$$
$$\mu = 0.012155092$$
$$t_f = 12.567475674$$

(4) Direct periodic orbit about m_2

$$x_0 = 1.399999991$$
$$\dot{y}_0 = -0.9298385561$$
$$\mu = 0.012155092$$
$$t_f = 13.775148738$$

The following is the graphic screen display for main menu option `<3>` and an integration step size of `0.01`. The locations of the Earth and Moon are labeled with an asterisk and the L_3 libration point is the small dot.



Another excellent source of circular-restricted three-body problems is described in the report, “A Collection of Restricted Three-body Test Problems” by Philip W. Sharp, Report Series 460, Department of Mathematics, University of Auckland, March 2001. A copy of this report in PDF format can be found at the following web site:

http://www.scitec.auckland.ac.nz/~sharp/papers/preprints.html

The following is a summary of these test problems. Column 1 is the test problem number, column 2 is the initial *x* coordinate x_0, column 3 is the initial *y* speed $\dot{y}_0$ and column 4 is the orbital period t_f. For problems 1 to 15 $\mu = 0.012277471$ and for problems 16 to 20 $\mu = 0.000953875$.

1	0.994000E+00	-0.21138987966945026683E+01	0.54367954392601899690E+01
2	0.994000E+00	-0.20317326295573368357E+01	0.11124340337266085135E+02
3	0.994000E+00	-0.20015851063790825224E+01	0.17065216560157962559E+02
4	0.997000E+00	-0.16251217072210773125E+01	0.22929723423442969481E+02
5	0.879962E+00	-0.66647197988564140807E+00	0.63006757422352314657E+01
6	0.879962E+00	-0.43965281709207999128E+00	0.12729711861022426544E+02
7	0.879962E+00	-0.38089067106386964470E+00	0.19138746281183026809E+02
8	0.997000E+00	-0.18445010489730401177E+01	0.12353901248612092736E+02
9	0.100000E+01	-0.16018768253456252603E+01	0.12294387796695023304E+02
10	0.100300E+01	-0.14465123738451062297E+01	0.12267904265603897140E+02
11	0.120000E+01	-0.71407169828407848921E+00	0.18337451820715063383E+02
12	0.120000E+01	-0.67985320356540547720E+00	0.30753758552146029263E+02
13	0.120000E+01	-0.67153130632829144331E+00	0.43214375227857454128E+02
14	0.120000E+01	-0.66998291305226832207E+00	0.55672334134347612727E+02
15	0.120000E+01	-0.66975741517271092087E+00	0.68127906604713772763E+02
16	-0.102745E+01	0.40334488290490413053E-01	0.18371316400018903965E+03
17	-0.976680E+00	-0.61191623926410837000E-01	0.17733241131524483004E+03
18	-0.766650E+00	-0.51230158665978820282E+00	0.17660722897242937108E+03
19	-0.109137E+01	0.14301959822238380020E+00	0.82949461922342093092E+02
20	-0.110137E+01	0.15354250908611454510E+00	0.60952121909407746612E+02

zvcurve1.m – graphics display of zero velocity curves through equilibrium points

This MATLAB script can be used to display zero relative velocity curves that pass through the five equilibrium or libration points of the Earth-Moon circular-restricted three-body problem (CRTBP). These two-dimensional drawings are also called *equipotential* curves. The zero velocity curves define regions of the *x-y* plane that a satellite or celestial body with a given energy level can or cannot attain.

If we set the velocity components to zero in the equation for the Jacobi integral, we find that the Jacobi constant can be determined from the following expression:

$$C = -2E_0 = (1-\mu)\left(r_1^2 + \frac{2}{r_1}\right) + \mu\left(r_2^2 + \frac{2}{r_2}\right) - \mu(1-\mu)$$

where E_0 is the zero relative velocity *specific orbital energy* which is given by.

$$E_0 = -\frac{1}{2}\left(x^2 + y^2\right) - \frac{1-\mu}{r_1} - \frac{\mu}{r_2}$$

From the Jacobi integral we can see that imaginary velocities occur whenever the following expression is smaller than the Jacobi constant *C*.

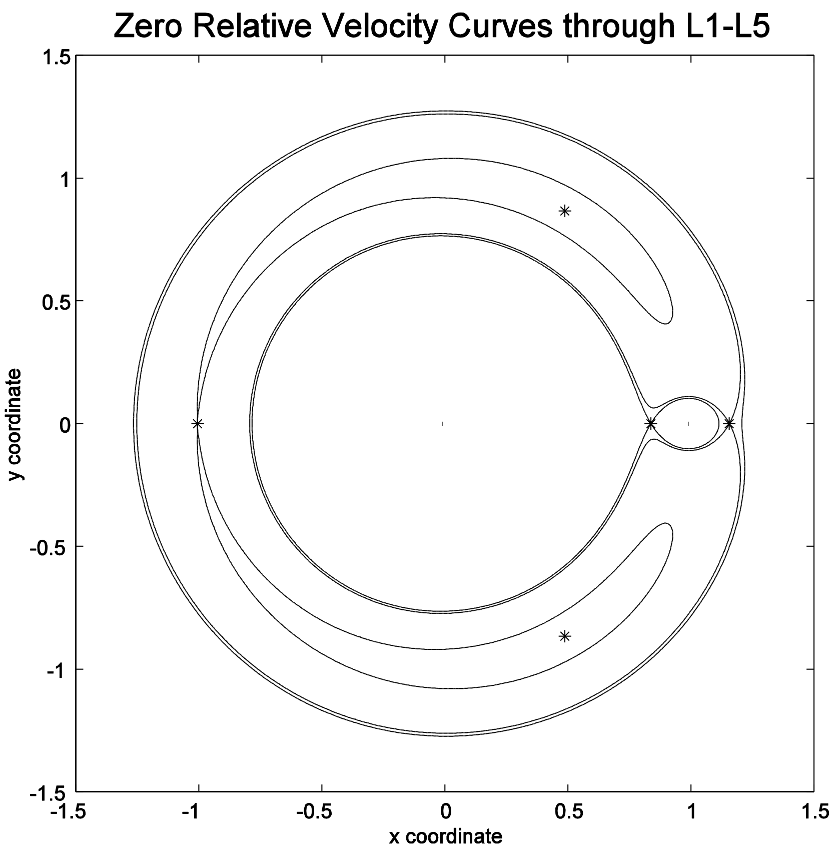
$$x^2 + y^2 + \frac{2(1-\mu)}{r_1} + \frac{2\mu}{r_2}$$

This MATLAB script computes and displays zero relative velocity contour plots that actually pass through the five libration points.

The following is a typical user interaction with this script.

```
                program zvcurve1

   graphics display of zero relative velocity
   curves through the crtbp equilibrium points


    <1> L1 contour

    <2> L2 contour

    <3> L3 contour

    <4> L4 and L5 contour

    <5> all points contour

    selection (1, 2, 3, 4 or 5)

   ?5
```

The following is a graphics display of the equipotential curves through each libration point. The libration points are labeled with asterisks and the Earth and Moon are shown as small dots.



zvcurve2.m – graphics display of user-defined zero relative velocity curves

This MATLAB script can be used to display zero relative velocity contour curves of the circular-restricted three-body problem (CRTBP) for user-defined values of the Jacobi constant *C* and mass ratio μ as defined by the following expression:

$$C = (1-\mu)\left(r_1^2 + \frac{2}{r_1}\right) + \mu\left(r_2^2 + \frac{2}{r_2}\right) - \mu(1-\mu)$$

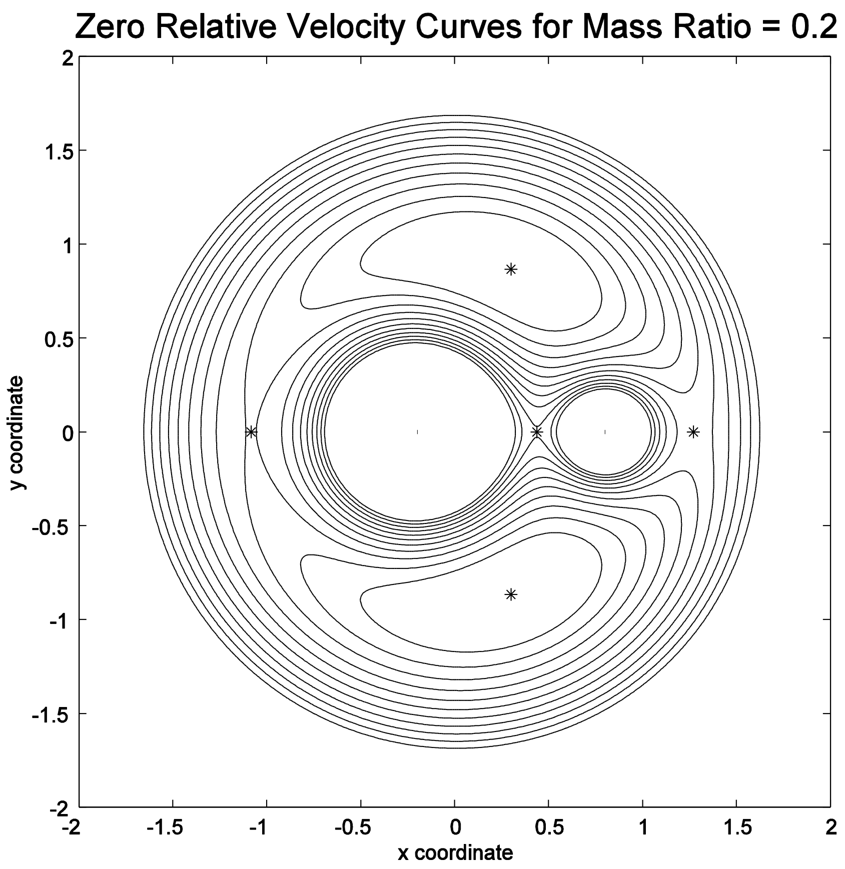
This application will prompt the user for the mass ratio and the contour levels to plot either as discrete values or as a range of values with a user-defined increment. The software will also ask you for the minimum and maximum values of the *x* and *y* coordinates and an increment for creating the mesh grid used by MATLAB to plot the contours. Please note that if the mesh grid increment is too large the script may not plot “complete” contour levels.

The following is a typical user interaction with this MATLAB script.

```
            program zvcurve2

      graphics display of user-defined
       zero relative velocity curves
```

```
please input the value for the mass ratio
? 0.2


      contour level input menu

 <1> lower value, increment, upper value

 <2> discrete contour values

 selection (1 or 2)

? 1


please input the lower contour value
? 3.0

please input the contour increment
? 0.1

please input the upper contour value
? 4.0


would you like to label the contours (y = yes, n = no)
? n


please input the minimum value for the x and y coordinates
? -2

please input the maximum value for the x and y coordinates
? 2

please input the mesh grid value for x and y
(a value between 0.01 and 0.005 is recommended)
? 0.01
```

The following is the graphics display for this example. The libration points for this three-body system are labeled with asterisks and the primary and secondary bodies are shown as small dots.

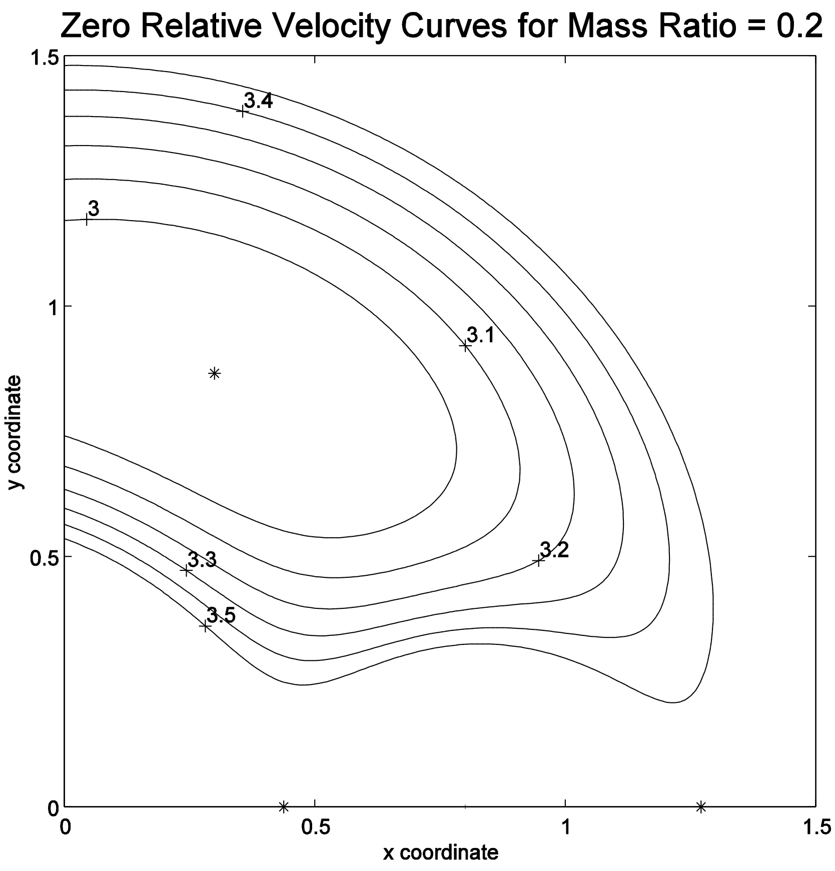


The software also allows the user to control what part of the contour plot is displayed and label the zero relative velocity contours. The following is a typical user interaction illustrating these script options.

```
         program zvcurve2

   graphics display of user-defined
    zero relative velocity curves


   please input the value for the mass ratio
   ? 0.2


         contour level input menu

    <1> lower value, increment, upper value

    <2> discrete contour values

    selection (1 or 2)

   ? 1


   please input the lower contour value
   ? 3.0
```

```
please input the contour increment
? 0.1

please input the upper contour value
? 3.5


would you like to label the contours (y = yes, n = no)
? y


please input the minimum value for the x and y coordinates
? 0

please input the maximum value for the x and y coordinates
? 1.5

please input the mesh grid value for x and y
(a value between 0.01 and 0.005 is recommended)
? 0.01
```

The following is the graphics display for this example. The libration points for this three-body system are labeled with asterisks and the primary and secondary bodies are shown as small dots.



11. Apparent Coordinates of a Planet or the Moon

These two MATLAB functions can be used to determine the apparent geocentric and topocentric coordinates of a planet or the Moon. These functions use several functions ported to MATLAB using the Fortran versions of the `NOVAS` (Naval Observatory Vector Astrometry Subroutines) source code that was developed at the United States Naval Observatory.

These functions includes the following types of coordinate corrections:

(1) precession

(2) nutation

(3) aberration

(4) gravitational deflection of light

Please note that these functions do not include a correction for atmospheric refraction.

An excellent discussion about these corrections can be found in “Mean and Apparent Place Computations in the New IAU System. III. Apparent, Topocentric, and Astrometric Places of Planets and Stars”, G.H. Kaplan, J.A. Hughes, P.K. Seidelmann, C.A. Smith and B.D. Yallop, *The Astronomical Journal*, Vol. 97, No. 4, pages 1197-1210, 1989.

The processing steps used in these two MATLAB scripts are as follows:

(1) input a calendar date on the UT1 time scale

(2) compute the UT1 Julian date

(3) compute the TDT Julian date

(4) compute the TDB Julian date

(5) select the target body

(6) define type of coordinates

(7) evaluate the ephemeris

The topocentric azimuth A and elevation (altitude) h are calculated using the topocentric right ascension α and declination δ and the observer's coordinates as follow:

$$\tan A = \frac{\sin H}{\cos H \sin\phi - \tan\delta \cos\phi}$$

where H is the *apparent hour angle* of the celestial object and ϕ is the geodetic latitude of the observer. The apparent hour angle is given by the difference between the observer's local sidereal time θ and the right ascension of the object as follows:

$$H = \theta - \alpha$$

In these MATLAB functions azimuth is measured positive clockwise from north. For example, east is an azimuth of 90°, south is 180° and so forth.

function applan1.m

The source ephemeris for this routine is the DE405 binary file `jpleph`.

The following is the syntax for this MATLAB function.

```
function [ra, dec, dis] = applan1 (tjd, ujd, l, n, topo, glon, glat, ht)

% this function computes the apparent geocentric or topocentric place
% of a planet or other solar system body. rectangular coordinates of
% solar system bodies are obtained from function solsys.

% input

%  tjd  = tdt julian date for apparent geocentric place
%  ujd  = ut1 julian date for apparent topocentric place
%  l    = body identification number for desired planet
%  n    = body identification number for the earth
%  topo = type of apparent place calculation
%       = 0 ==> geocentric
%       = 1 ==> topocentric
%  glon = geodetic longitude of observer (east +, degrees)
%  glat = geodetic latitude of observer (north +, degrees)
%  ht   = height of observer (meters)

% output

%  ra  = apparent geocentric or topocentric right ascension,
%        referred to true equator and equinox of date (hours)
%  dec = apparent geocentric or topocentric declination,
%        referred to true equator and equinox of date (degrees)
%  dis = true distance from earth to planet (astronomical units)
```

A MATLAB script called `aplanet1` demonstrates how to interact with this function. This script will prompt you for the calendar date, universal time, observer coordinates, the “target” body and the type of ephemeris (topocentric or geocentric). The program defaults to the Earth as the central or “primary” body.

The following is a typical user interaction with this demonstration script.

```
      aplanet1 - apparent coordinates


   please input a UTC calendar date

   please input the calendar date
```

```
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,1998

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

    target body menu

  <1> Mercury
  <2> Venus
  <3> Earth
  <4> Mars
  <5> Jupiter
  <6> Saturn
  <7> Uranus
  <8> Neptune
  <9> Pluto
  <10> Sun
  <11> Moon

please select the target body
? 11

coordinate type menu

 <1> geocentric

 <2> topocentric

please select coordinate type
? 2

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 40,0,0

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -105,0,0

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 1000
```

The following is the program output created for this example.

```
 apparent topocentric coordinates

    'Moon'

UTC calendar date       01-Jan-1998

universal time          10:20:30

UTC julian date          2450814.9309

observer latitude        -105d 00m 00.00s

observer east longitude  +40d 00m 00.00s
observer altitude           1000.0000 meters

right ascension          +21h 12m 2.9587s

declination              -14d 17m 47.57s

azimuth                  +27d 31m 13.82s

elevation                -61d 53m 39.08s

topocentric distance      376284.4187 kilometers
```

function applan2.m

The source ephemeris for this routine is the SLP96 binary file `slp96`.

The following is the syntax for this function.

```
function [ra, dec, dis] = applan2(tjd, ujd, l, n, topo, glon, glat, ht)

% this function computes the apparent geocentric or topocentric place
% of a planet or other solar system body. rectangular coordinates of
% solar system bodies are obtained from function slp96.

% input

%  tjd  = tdt julian date for apparent geocentric place
%  ujd  = ut1 julian date for apparent topocentric place
%  l    = body identification number for desired planet
%  n    = body identification number for the earth
%  topo = type of apparent place calculation
%       = 0 ==> geocentric
%       = 1 ==> topocentric
%  glon = geodetic longitude of observer (east +, degrees)
%  glat = geodetic latitude of observer (north +, degrees)
%  ht   = height of observer (meters)

% output

%  ra  = apparent geocentric or topocentric right ascension,
%        referred to true equator and equinox of date (hours)
%  dec = apparent geocentric or topocentric declination,
%        referred to true equator and equinox of date (degrees)
%  dis = true distance from earth to planet (astronomical units)
```

The MATLAB script called `aplanet2` demonstrates how to use this function. The user interaction with this script is identical to the interaction with the `aplanet1` script described above.

12. Apparent Coordinates of a Star

These two MATLAB scripts demonstrate how to calculate the apparent topocentric and geocentric coordinates of a star. The first script is called `astar1` and it uses DE405 as its source ephemeris. The second script is called `astar2` and it uses SLP96 as its source ephemeris. These applications use several functions ported to MATLAB from the Fortran versions of the NOVAS (Naval Observatory Vector Astrometry Subroutines) source code that was developed at the United States Naval Observatory.

This program includes the following types of coordinate corrections:

(1) precession

(2) nutation

(3) aberration

(4) gravitational deflection of light

(5) proper motion

(6) parallax

(7) radial velocity

Please note that this program does not include a correction for atmospheric refraction. For extragalatic objects the proper motions, parallax and radial velocity should all be set to 0.

An excellent discussion about these corrections can be found in “Mean and Apparent Place Computations in the New IAU System. III. Apparent, Topocentric, and Astrometric Places of Planets and Stars”, G.H. Kaplan, J.A. Hughes, P.K. Seidelmann, C.A. Smith and B.D. Yallop, *The Astronomical Journal*, Vol. 97, No. 4, pages 1197-1210, 1989.

The processing steps used in this program are as follows:

(1) input a calendar date on the UT1 time scale

(2) compute the UT1 Julian date

(3) compute the TDT Julian date

(4) compute the TDB Julian date

(5) read the star's coordinates, proper motions, parallax and radial velocity

(6) define type of coordinates

(7) calculate apparent coordinates

The syntax and arguments for the MATLAB function that actually performs most of the calculations are as follows:

```
function [ra, dec] = apstar1 (tjd, ujd, n, topo, glon, glat, ht, ...
                              ram, decm, pmra, pmdec, parlax, radvel)

% this subroutine computes the geocentric or topocentric apparent place
% of a star, given its mean place, proper motion, parallax, and radial
% velocity for j2000.0.

% input

%  tjd    = tdt julian date for apparent place
%  ujd    = ut1 julian date for apparent topocentric place
%  n      = body identification number for the earth
%  topo   = type of apparent place calculation
%         = 0 ==> geocentric
%         = 1 ==> topocentric
%  ram    = mean right ascension j2000.0 (hours)
%  decm   = mean declination j2000.0 (degrees)
%  pmra   = proper motion in ra (seconds of time/julian century)
%  pmdec  = proper motion in dec (seconds of arc/julian century)
%  parlax = parallax (seconds of arc)
%  radvel = radial velocity (kilometers per second)

% output

%  ra  = apparent geocentric or topocentric right ascension,
%        referred to true equator and equinox of date (hours)
%  dec = apparent geocentric or topocentric declination,
%        referred to true equator and equinox of date (degrees)
```

The `astar` software will prompt you for the calendar date, universal time, observer coordinates, the name of the star data file and the type of ephemeris (topocentric or geocentric). The following is a typical star data file named `altair.dat`. Do not delete any lines of text or data as the software expects to find exactly 20 lines of information.

```
star name
ALTAIR

J2000 right ascension (hours)
19.8463894440

J2000 declination (degrees)
8.8683416670

J2000 proper motion in right ascension (seconds/Julian century)
3.6290

J2000 proper motion in declination (arcseconds/Julian century)
38.6300

parallax (arcseconds)
0.1981

radial velocity (kilometers/second)
-26.30
```

The following is a typical user interaction with the `star1` (and `star2`) scripts. It illustrates the topocentric coordinates of the star Altair relative to a sea level observer at 40° north latitude and 105° west longitude, at 0 hours UT.

```
star1 - apparent coordinates - jplephem ephemeris

please input a UTC calendar date

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

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


please input the star data file name
(be sure to include the file name extension)
? altair.dat

coordinate type menu

 <1> geocentric

 <2> topocentric

please select coordinate type
? 2

please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 40,0,0

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -105,0,0

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 0


 apparent topocentric coordinates

ALTAIR

UTC calendar date        01-Jan-1998

universal time           00:00:00

UTC julian date          2450814.5000

observer latitude        -105d 00m 00.00s

observer east longitude  +40d 00m 00.00s

observer altitude              0.0000 meters

right ascension          +19h 50m 39.1674s

declination              +08d 51m 46.09s
```

13. JPL DE405 Lunar, Solar and Planetary Ephemeris

This MATLAB script demonstrates how to read DE405 binary ephemeris files created using the Fortran programs and ASCII data files provided by the Jet Propulsion Laboratory, and calculate the position and velocity vectors of a planet or the Moon. This function was ported to MATLAB using the Fortran source code subroutine provided by JPL. These ASCII files can be found on the Internet at **ftp://navigator.jpl.nasa.gov**. A CD ROM containing these ASCII data files and Fortran source code is also available from Willmann-Bell.

The `jplephem` function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
iephem = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function.

`Demojpl` is a simple MATLAB script that demonstrates how to call the `jplephem` routine. A sample binary file for IBM-PC compatible computers is included on the *Celestial Computing with MATLAB* CD ROM. The name of this binary file is `jpleph` and it is valid between Julian date 2429616.5 (calendar date 12/19/1939) and Julian date 2473488.5 (calendar date 1/30/2060). The coordinates calculated by this function are with respect to the International Celestial Reference System (ICRS) which is described in "The International Celestial Reference Frame as Realized by Very Long Baseline Interferometry", C. Ma, E. Arias, T. Eubanks, A. Fey, A. Gontier, C. Jacobs, O. Sovers, B. Archinal and P. Charlot, *The Astronomical Journal*, 116:516-546, 1998, July.

The following is the syntax for this MATLAB function:

```
function rrd = jplephem (et, ntarg, ncent)

% reads the jpl planetary ephemeris and gives
% the position and velocity of the point 'ntarg'
% with respect to point 'ncent'

% input

%   et    = julian ephemeris date at which interpolation is wanted

%   ntarg = integer number of 'target' point

%   ncent = integer number of center point

%   the numbering convention for 'ntarg' and 'ncent' is:

%         1 = mercury           8 = neptune
%         2 = venus             9 = pluto
%         3 = earth            10 = moon
%         4 = mars             11 = sun
%         5 = jupiter          12 = solar-system barycenter
%         6 = saturn           13 = earth-moon barycenter
%         7 = uranus           14 = nutations (longitude and obliq)
%                            15 = librations, if on ephemeris file

%        if nutations are wanted, set ntarg = 14.
%        for librations, set ntarg = 15. set ncent = 0.

% output

%   rrd = output 6-word array containing position and velocity
%         of point 'ntarg' relative to 'ncent'. the units are au and
%         au/day. for librations the units are radians and radians
%         per day. in the case of nutations the first four words of
%         rrd will be set to nutations and rates, having units of
%         radians and radians/day.

%         the option is available to have the units in km and km/sec.
%         for this, set km = 1 via global in the calling program.
```

The following is a typical user interaction with this script.

```
   demojpl - demonstrates how to use the jplephem.m function


please input a UTC calendar date

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 10,21,1998

    target body menu

  <1> Mercury
  <2> Venus
  <3> Earth
  <4> Mars
  <5> Jupiter
  <6> Saturn
  <7> Uranus
  <8> Neptune
  <9> Pluto
  <10> Moon
  <11> Sun

please select the target body
? 10

    central body menu

  <1> Mercury
  <2> Venus
  <3> Earth
  <4> Mars
  <5> Jupiter
  <6> Saturn
  <7> Uranus
  <8> Neptune
  <9> Pluto
  <10> Moon
  <11> Sun
  <12> solar-system barycenter
  <13> Earth-Moon barycenter

please select the central body
```

```
? 3

target body        'Moon'

central body       'Earth'

UTC calendar date       21-Oct-1998

UTC Julian date          2451107.5000

state vector

      rx (km)             ry (km)             rz (km)             rmag (km)
 -3.3736590217e+005  -2.1902562142e+005  -5.9817990536e+004  +4.0665239060e+005

      vx (kps)            vy (kps)            vz (kps)            vmag (kps)
 +5.3908644803e-001  -7.5535385071e-001  -2.8617936544e-001  +9.7111907995e-001
```

14. SLP96 Lunar, Solar and Planetary Ephemeris

This MATLAB function demonstrates how to read SLP96 binary ephemeris files created using the Fortran programs and ASCII data files provided by the Bureau of Longitudes (BDL, Paris), and calculate the position and velocity vectors of a planet or the moon in either the J2000 FK5 or dynamical coordinate systems. This function was ported to MATLAB using the Fortran source code subroutine provided by the BDL. These ASCII files can be found on the Internet at **ftp.bdl.fr/pub/ephem/sun/slp96**.

The `slp96` function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
islp96 = 1;
```

The user must also specify the name of the binary data file with the following statement:

```
fname = ‘slp96’
```

These variables should also be placed in a `global` statement at the beginning of the main script that calls this function.

`Demoslp` is a simple MATLAB driver script that demonstrates how to call the `slp96` routine. A sample binary file for IBM-PC compatible computers is included on the *Celestial Computing with MATLAB* CD ROM. This binary file is valid between Julian date 2433264.5 (calendar date 12/14/1949) and Julian date 2469808.5 (calendar date 1/02/2050).

The following is a summary of the inputs and outputs for this MATLAB function:

```
function [result, ierr] = slp96 (tjd, ibody, icent, ipv, iframe, icoord)

% solar, lunar and planetary coordinates j2000

% input

%     tjd       julian date tdb

%     ibody     body index

%               ibody=01 : mercury       ibody=07 : uranus
%               ibody=02 : venus         ibody=08 : neptune
%               ibody=03 : e-m baryc.    ibody=09 : void
%               ibody=04 : mars          ibody=10 : moon
%               ibody=05 : jupiter       ibody=11 : sun
%               ibody=06 : saturn        ibody=12 : earth

%     icent     frame center index

%               icent=00 : barycenter of solar system
%               icent=01 : mercury       icent=07 : uranus
%               icent=02 : venus         icent=08 : neptune
%               icent=03 : e-m baryc.    icent=09 : void
%               icent=04 : mars          icent=10 : moon
%               icent=05 : jupiter       icent=11 : sun
%               icent=06 : saturn        icent=12 : earth

%     ipv       position-velocity index

%               ipv=1 : position.
%               ipv=2 : position and velocity.

%     iframe    frame index

%               iframe=1 : equinox and equator  j2000 (fk5)
%               iframe=2 : equinox and ecliptic j2000 (dynamical)

%     icoord    coordinates index

%               icoord=1 : rectangular coordinates
%               icoord=2 : spherical coordinates

%     fname     name of the slp96 data file
%               note: passed via global

% output

%     result    results table

%               rectangular coordinates (icoord = 1)

%               position

%               result(1) : equatorial or ecliptic component x (au)
%               result(2) : equatorial or ecliptic component y (au)
%               result(3) : equatorial or ecliptic component z (au)

%               velocity

%               result(4) : equatorial or ecliptic component x (au/day)
%               result(5) : equatorial or ecliptic component y (au/day)
%               result(6) : equatorial or ecliptic component z (au/day)

%               spherical coordinates (icoord = 2)

%               position

%               result(1) : right ascension or longitude (radians)
%               result(2) : declination or latitude (radians)
%               result(3) : geometric distance (au)

%               velocity

%               result(4) : right ascension or longitude (rad/day)
%               result(5) : declination or latitude (rad/day)
%               result(6) : geometric distance (au/day)

%     ierr      error index

%               ierr=0  : no error
%               ierr=10 : file error
%               ierr=11 : date error (tjd)
%               ierr=12 : body error (ibody)
%               ierr=13 : frame center error (icent)
%               ierr=14 : position-velocity error (ipv)
%               ierr=15 : frame error (iframe)
%               ierr=16 : coordinates error (icoord)
```

The processing steps used in this script are as follows:

(1) input a calendar date on the UTC time scale

(2) compute the UTC Julian date

(3) compute the TDT Julian date

(4) compute the TDB Julian date

(5) select the target body

(6) select the central body

(7) define type of coordinate system

(8) define type of coordinate calculations

(9) specify the name of the SLP96 binary data file

(10) evaluate the ephemeris

The following is a typical user interaction with this script.

```
   demoslp - demonstrates how to use the slp96.m function


please input a UTC calendar date

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 10,21,1998

    target body menu

  <1> Mercury
  <2> Venus
  <3> Earth-Moon barycenter
  <4> Mars
  <5> Jupiter
  <6> Saturn
  <7> Uranus
  <8> Neptune
  <9> void
  <10> Moon
  <11> Sun
  <12> Earth

please select the target body
? 10

    central body menu

  <0> solar system barycenter
  <1> Mercury
  <2> Venus
  <3> Earth-Moon barycenter
  <4> Mars
  <5> Jupiter
  <6> Saturn
  <7> Uranus
  <8> Neptune
  <9> void
  <10> Moon
  <11> Sun
  <12> Earth

please select the central body
? 12


    coordinate system menu

  <1> equinox and equator j2000 (fk5)

  <2> equinox and ecliptic j2000 (dynamical)

please select the coordinate system type
? 1


    coordinate type menu

  <1> rectangular coordinates

  <2> spherical coordinates

please select the coordinate type
? 1
```

The following is the program output created for this example.

```
program demoslp

equinox and equator j2000 (fk5)

target body        'Moon'

central body       'Earth'


UTC calendar date       21-Oct-1998

UTC Julian date          2451107.5000


state vector

      rx (km)              ry (km)              rz (km)              rmag (km)
-3.3736588697e+005  -2.1902566702e+005  -5.9818049020e+004  +4.0665241115e+005

      vx (kps)             vy (kps)             vz (kps)             vmag (kps)
 +5.3908659599e-001  -7.5535382805e-001  -2.8617931903e-001  +9.7111913078e-001
```

15. Other Lunar, Solar and Planetary Ephemeris Algorithms

The next ten MATLAB functions are based on the algorithms described in "Low-Precision Formulae for Planetary Positions", T. C. Van Flandern and K. F. Pulkkinen, *The Astrophysical Journal Supplement Series*, **41**:391-411, November 1979. To the precision of this algorithm (one arc minute) these coordinates can be considered to be true-of-date.

Each algorithm uses time arguments given by

$$t = JD - 2451545$$
$$T = t/36525 + 1$$

where *JD* is the Julian date on the UT1 time scale.

Each MATLAB function uses fundamental trigonometric arguments (in revolutions) of the following form:

$$\begin{aligned} G_s &= 0.993126 + 0.00273777850t \\ G_2 &= 0.140023 + 0.00445036173t \\ G_4 &= 0.053856 + 0.00145561327t \\ G_5 &= 0.056531 + 0.00023080893t \\ F_4 &= 0.849694 + 0.00145569465t \\ L_4 &= 0.987353 + 0.00145575328t \end{aligned}$$

The heliocentric, ecliptic longitude λ, latitude β and distance *r* are computed from series involving these arguments. These series are of the form

$$\begin{aligned} \lambda &= L_4 + 38451\sin G_4 + 2238\sin(2G_4) + 181\sin(3G_4) + \ldots \\ \beta &= 6603\sin F_4 + 622\sin(G_4 - F_4) + 615\sin(G_4 + F_4) + \ldots \\ r &= 1.53031 - 0.1417\cos G_4 - 0.0066\cos(2G_4) + \ldots \end{aligned}$$

where the unit of *r* is Astronomical Units.

The heliocentric, ecliptic position vector of the planet is determined from

$$\mathbf{r} = r\begin{Bmatrix} \cos\beta\cos\lambda \\ \cos\beta\sin\lambda \\ \sin\beta \end{Bmatrix}$$

sun.m – solar ephemeris

This function calculates the true-of-date geocentric right ascension, declination and position vector of the Sun.

The syntax of this MATLAB function is

```
function [rasc, decl, rsun] = sun (jdate)

% solar ephemeris

% input

%  jdate = Julian date

% output

%  rasc = right ascension of the sun (radians)
%         (0 <= rasc <= 2 pi)
%  decl = declination of the sun (radians)
%         (-pi/2 <= decl <= pi/2)
%  rsun = eci position vector of the sun (km)
```

moon.m – lunar ephemeris

This function calculates the true-of-date geocentric right ascension, declination and position vector of the Moon.

The syntax of this MATLAB function is

```
function [rasc, decl, rmoon] = moon (jdate)

% lunar ephemeris

% input

%  jdate = julian date

% output

%  rasc  = right ascension of the moon (radians)
%          (0 <= rasc <= 2 pi)
%  decl  = declination of the moon (radians)
%          (-pi/2 <= decl <= pi/2)
%  rmoon = eci position vector of the moon (km)
```

mercury.m – Mercury ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Mercury.

The syntax of this MATLAB function is

```
function rmercury = mercury (jdate)

% true-of-date heliocentric, ecliptic
% position vector of Mercury

% input

%  jdate = julian date
% output

%  rmercury = position vector of Mercury (km)
```

venus.m – Venus ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Venus.

The syntax of this MATLAB function is

```
function rvenus = venus (jdate)

% true-of-date heliocentric, ecliptic
% position vector of venus

% input

%  jdate = Julian date

% output

%  rvenus = position vector of Venus (km)
```

earth.m – Earth ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of the Earth.

The syntax of this MATLAB function is

```
function rearth = earth (jdate)

% true-of-date heliocentric, ecliptic
% position vector of the Earth

% input

%  jdate = Julian date

% output

%  rearth = position vector of the Earth (km)
```

mars.m – Mars ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Mars.

The syntax of this MATLAB function is

```
function rmars = mars (jdate)

% true-of-date heliocentric, ecliptic
% position vector of Mars

% input

%  jdate = Julian date
% output

%  rmars = position vector of Mars (km)
```

jupiter.m – Jupiter ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Jupiter.

The syntax of this MATLAB function is

```
function rjupiter = jupiter (jdate)

% true-of-date heliocentric, ecliptic
% position vector of Jupiter

% input

%  jdate = Julian date

% output

%  rjupiter = position vector of Jupiter (km)
```

saturn.m – Saturn ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Saturn.

The syntax of this MATLAB function is

```
function rsaturn = saturn (jdate)

% true-of-date heliocentric, ecliptic
% position vector of Saturn

% input

%  jdate = Julian date

% output

%  rsaturn = position vector of Saturn (km)
```

uranus.m – Uranus ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Uranus.

The syntax of this MATLAB function is

```
function ruranus = uranus (jdate)

% true-of-date heliocentric, ecliptic
% position vector of Uranus

% input
%  jdate = Julian date

% output

%  ruranus = position vector of Uranus (km)
```

neptune.m – Neptune ephemeris

This function calculates the true-of-date heliocentric ecliptic position vector of Neptune.

The syntax of this MATLAB function is

```
function rneptune = neptune (jdate)

% true-of-date heliocentric, ecliptic
% position vector of Neptune

% input

%  jdate = Julian date

% output

%  rneptune = position vector of Neptune (km)
```

pluto.m – Pluto ephemeris

This function calculates the heliocentric position vector of Pluto relative to the ecliptic and equinox of J2000. This algorithm is based on the method described in Chapter 36 of *Astronomical Algorithms* by Jean Meeus.

The fundamental time argument for this method is a function of the Julian Ephemeris Date *JED* as follows:

$$T = \frac{JED - 2451545}{36525}$$

The heliocentric ecliptic coordinates of Pluto are computed from series of the form

$$\lambda = \lambda^m + \sum_{i=1}^{43} A\sin\alpha + B\cos\alpha$$

where

$\lambda^m =$ coordinate mean value

$\alpha = iJ + jS + kP$

$J, S, P =$ mean longitudes of Jupiter, Saturn and Pluto

$i, j, k =$ integer constants

$A, B =$ coefficients of periodic term

The syntax of this MATLAB function is

```
function rpluto = pluto(jdate)

% heliocentric coordinates of pluto

% heliocentric position vector of Pluto
% ecliptic and equinox of J2000

% input

%  jdate = Julian date

% output

%  rpluto = position vector of Pluto (km)
```

sun1.m – precision Sun ephemeris

This MATLAB function computes a true-of-date geocentric ephemeris of the Sun based on the data and numerical methods described in the book, *Planetary Programs and Tables* by Pierre Bretagnon and Jean-Louis Simon.

The fundamental time argument of this method is the number of days relative to the Julian epoch January 1, 2000 normalized with respect to 3652500 Julian days. This value can be calculated for any Julian Ephemeris Date *JED* with the following expression

$$U = \frac{JED - 2451545}{3652500}$$

The geocentric, ecliptic *mean* longitude of the Sun is calculated with a trigonometric series of the form

$$\lambda_s^m = \lambda_0 + \lambda_1 U + \sum_{i=1}^{50} l_i \sin\left(\alpha_i + \nu_i U\right)$$

The geocentric distance of the Sun is calculated with another series of the form

$$r_s = r_0 + r_1 U + \sum_{i=1}^{50} r_i \cos\left(\alpha_i + \nu_i U\right)$$

The longitude of the Sun is corrected for the effect of *aberration* (in radians) with the following equation:

$$\Delta\lambda_s^a = 10^{-7}\left\{-993 + 17\cos\left(3.10 + 62830.14U\right)\right\}$$

The nutation in longitude (in radians) is calculated from

$$\Delta\psi = 10^{-7}\left(-834\sin A_1 - 64\sin A_2\right)$$

where

$$A_1 = 2.18 - 3375.70U + 0.36U^2$$
$$A_2 = 3.51 + 125666.39U + 0.10U^2$$

The apparent, geocentric ecliptic longitude of the Sun is determined as the combination of these three components with the next equation

$$\lambda_s = \lambda_s^m + \Delta\lambda_s^a + \Delta\psi$$

The three components of the geocentric, ecliptic position vector of the Sun are given by

$$x_s = r_s \cos\lambda_s$$
$$y_s = r_s \sin\lambda_s$$
$$z_s = 0$$

The apparent geocentric, equatorial right ascension α_s and declination δ_s of the Sun can be found from

$$\alpha_s = \arctan\left(\cos\lambda_s, \sin\varepsilon \sin\lambda_s\right)$$
$$\delta_s = \arcsin\left(\sin\varepsilon \sin\lambda_s\right)$$

where ε is the true obliquity of the ecliptic. This number is calculated from the mean obliquity of the ecliptic ε_m and the nutation in obliquity $\Delta\varepsilon$ in this function with the following expressions:

$$\varepsilon = \varepsilon_m + \Delta\varepsilon$$
$$\varepsilon_m = 10^{-7}\left(4090928 + 446\cos A_1 + 28\cos A_2\right)$$
$$\Delta\varepsilon = 10^{-7}U\left(-226938 + U\left(-75 + U\left(96926 + U\left(-2491 - 12104U\right)\right)\right)\right)$$

Finally, we can compute the three components of the *apparent*, geocentric equatorial position vector of the Sun with the following three expressions:

$$r_x = r\cos\alpha_s \cos\delta_s$$
$$r_y = r\sin\alpha_s \cos\delta_s$$
$$r_z = r\sin\delta_s$$

where *r* is the geocentric distance of the Sun.

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
suncoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function.

The syntax of this MATLAB function is

```
function [rasc, decl, rsun] = sun1 (jdate)

% precision ephemeris of the Sun

% input

%  jdate = julian ephemeris date

% output

%  rasc = right ascension of the Sun (radians)
%         (0 <= rasc <= 2 pi)
%  decl = declination of the Sun (radians)
%         (-pi/2 <= decl <= pi/2)
%  rsun = eci position vector of the Sun (km)
```

elp2000.m – osculating orbital elements of the moon

This function calculates the osculating classical orbital elements of the Moon in the mean ecliptic and mean equinox of date coordinate system. It is based on the book *Lunar Tables and Programs From 4000 B.C. TO A.D. 8000* by Michelle Chapront-Touze and Jean Chapront. This book and its optional companion software are available from Willmann-Bell **(www.willbell.com)**.

The fundamental time argument of this algorithm is

$$t = \frac{JED - 2451545}{36525}$$

where *JED* is the Julian Ephemeris Date.

The osculating orbital elements are calculated from series of the form

$$a = 383397.6 + S_a + tS''_a$$

where

$$S_a = \sum_{n=1}^{30} a_n \cos\left(\xi_n^{(0)} + \xi_n^{(1)} t + \xi_n^{(2)} 10^{-4} t^2 + \xi_n^{(3)} 10^{-6} t^3 + \xi_n^{(4)} 10^{-8} t^4\right)$$

$$S''_a = \sum_{n=1}^{3} a''_n \cos\left(\xi_n''^{(0)} + \xi_n''^{(1)} t\right)$$

The first few multipliers and trigonometric arguments for this orbital element are

n	a_n	$\xi_n^{(0)}$	$\xi_n^{(1)}$	$\xi_n^{(2)}$	$\xi_n^{(3)}$	$\xi_n^{(4)}$
1	3400.4	235.7004	890534.2230	-32.601	3.664	-1.769
2	-635.6	100.7370	413335.3554	-122.571	-10.684	5.028

n	a''_n	$\xi''^{(0)}_n$	$\xi''^{(1)}_n$
1	-0.55	238.2	854535.2
2	0.10	103.2	377336.3

The following is the syntax for this MATLAB function.

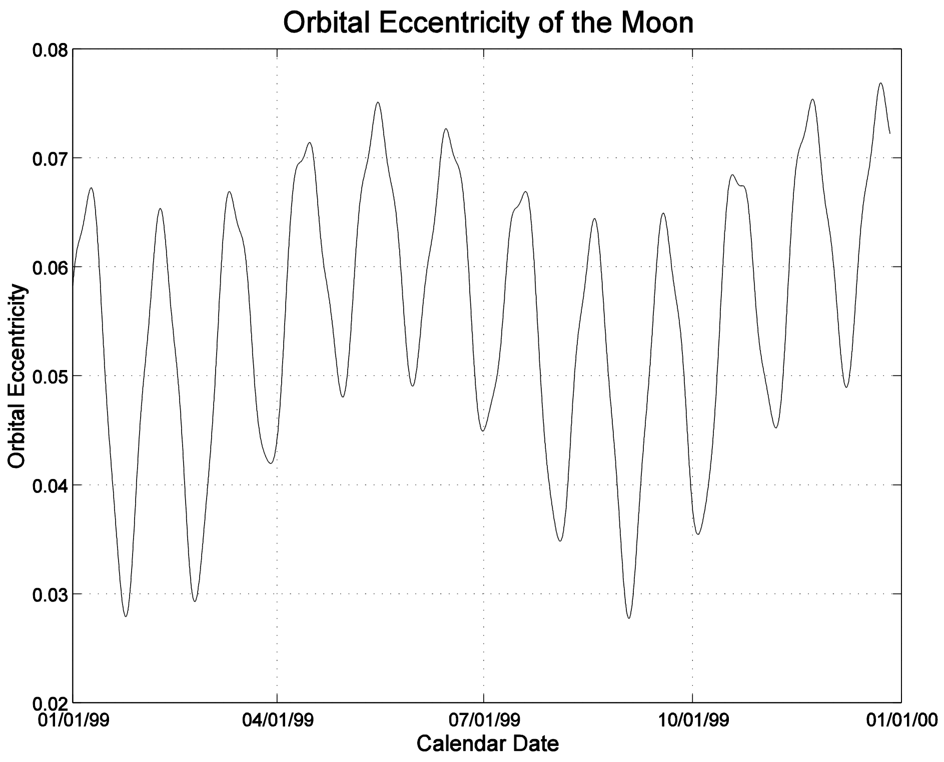
```
function oev = elp2000(tjd)

% osculating orbital elements of the moon

% mean ecliptic and mean equinox of date

% input

%  tjd = tdt Julian date

% 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)
```

A MATLAB script called `demoelp.m` which demonstrates how to interact with this ephemeris function is included with this software. The following is a typical user interaction with this script.

```
               demoelp

   < orbital elements of the moon >


please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,1999


please input the simulation period (days)
? 360
```

```
please input the graphics step size (days)
? .1

please select the orbital element to plot

  <1> semimajor axis
  <2> eccentricity
  <3> orbital inclination
  <4> argument of perigee
  <5> right ascension of the ascending node
  <6> true anomaly
  <7> apogee radius
  <8> perigee radius
  <9> geocentric distance

? 2
```

The following is the graphics display for this example.



planet1.m – mean ecliptic and equinox of data planetary ephemeredes

This MATLAB calculates the position and velocity vectors of the planets with respect to the mean ecliptic and equinox of date. These calculations are based on the algorithm described in Chapter 30 of *Astronomical Algorithms* by Jean Meeus. Each orbital element is represented by a cubic polynomial of the form

$$a_0 + a_1 T + a_2 T^2 + a_3 T^3$$

where the fundamental time argument *T* is given by

$$T = \frac{JED - 2451545}{36525}$$

In this expression *JED* is the Julian ephemeris date.

The syntax of this MATLAB function is

```
function [r, v] = planet1(ip, jdate)

% planetary ephemeris

% input

%  ip    = planet index (1 = mercury, 2 = venus, etc.)
%  jdate = Julian date

% output

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

planet2.m – mean ecliptic and equinox of J2000 planetary ephemerides

This MATLAB calculates the position and velocity vectors of the planets with respect to the mean ecliptic and equinox of J2000. These calculations are based on the algorithm described in Chapter 30 of *Astronomical Algorithms* by Jean Meeus. Each orbital element is represented by a cubic polynomial of the form

$$a_0 + a_1 T + a_2 T^2 + a_3 T^3$$

where the fundamental time argument *T* is given by

$$T = \frac{JED - 2451545}{36525}$$

In this expression *JED* is the Julian ephemeris date.

The syntax of this MATLAB function is

```
function [r, v] = planet2(ip, jdate)

% planetary ephemeris

% input

%  ip    = planet index (1 = mercury, 2 = venus, etc.)
%  jdate = Julian date
% output

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

seleno.m – selenographic coordinate transformation

This MATLB function computes a transformation matrix that can be used to convert vectors in the ECI mean of date coordinates to selenographic mean of date or true of date coordinate system. The transformation is based on the algorithms given in Section 4.8 of the classic text *Methods of Orbit Determination*.

The syntax of this MATLAB function is

```
function a = seleno(jd, it)

% transformation matrix from eci mean of date to
% selenographic mean of date or true of date

% input

%  jd = julian date
%  it = transformation flag
%     = 0 transform to mean selenographic
%    <> 0 transform to true selenographic
%         (include physical librations)

% output

%  a(3, 3) = transformation matrix
```

A MATLAB script that demonstrates how to use this function is included in the *Celestial Computing with MATLAB* software suite. The script is called `demosen.m` and it graphically displays the topocentric coordinates of the Sun relative to a selenographic location on the Moon.

16. Apparent Greenwich Sidereal Time

This section describes three MATLAB functions that can be used to calculate apparent Greenwich sidereal time.

gast.m – 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

$$\theta_m = 100.46061837 + 36000.770053608T + 0.000387933T^2 - T^3 / 38710000$$

where $T = (JD - 2451545)/36525$ and *JD* is the Julian date on the UT1 time scale. 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 nutations in obliquity and longitude involve the following three trigonometric arguments (in degrees):

$$L = 280.4665 + 36000.7698T$$

$$L' = 218.3165 + 481267.8813T$$

$$\Omega = 125.04452 - 1934.136261T$$

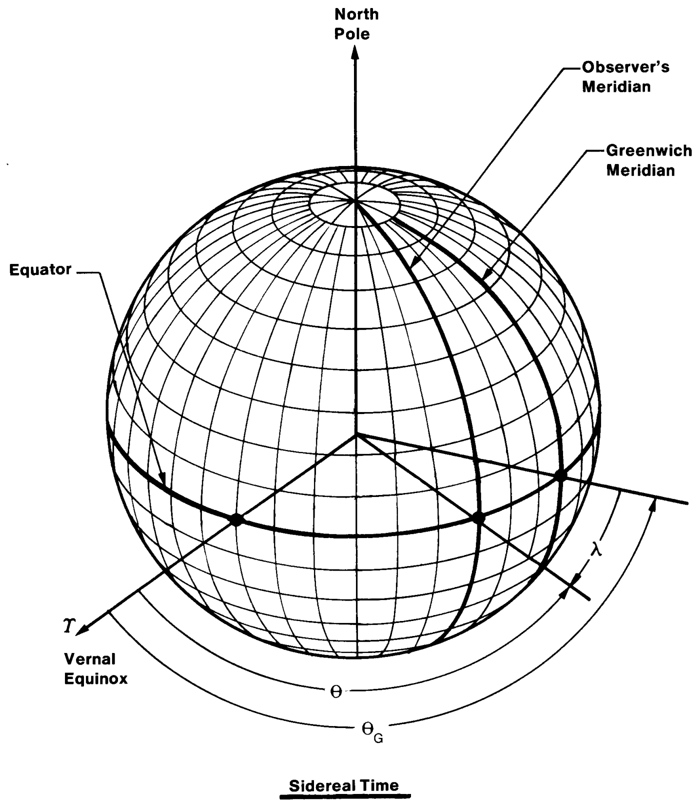
The calculation of the nutations use the following two equations:

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

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

where these corrections are in units of arc seconds.

The following diagram illustrates the geometry of sidereal time. In this picture λ is the observer's *west* longitude and θ is the observer's local sidereal time.



The syntax of this MATLAB function is as follows:

```
function gst = gast (jdate)

% Greenwich apparent sidereal time

% input

%  jdate = Julian date

% output

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

gast1.m – full precision

This function calculates the mean or apparent Greenwich sidereal time. For the apparent sidereal time calculation, the obliquity in longitude and obliquity are determined using the `nutation` 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` source 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 as follows:

```
function gst = gast1 (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
```

gast2.m – DE405 nutations

This function calculates the mean or apparent Greenwich sidereal time. For the apparent sidereal time calculation, the obliquity in longitude and obliquity are determined by reading the nutation values contained on a JPL DE405 binary ephemeris file named `jpleph`. 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` (Naval Observatory Vector Astrometry Subroutines) source code that was developed at the United States Naval Observatory.

The syntax of this MATLAB function is as follows:

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

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

% this version reads the nutation values from jpleph

% 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
```

17. IAU 1980 Nutation in Longitude and Obliquity

This MATLAB function (`nutation.m`) calculates the nutation in longitude and obliquity using the coefficients defined by the IAU 1980 nutation theory.

The nutation in longitude is determined from

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

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 number of terms in this nutation theory is 106.

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

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

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 $T = (JD - 2451545.0)/36525$ and *JD* is the Julian Date on the UT1 time scale.

If second-order terms are neglected, a *linearized* nutation matrix can be calculated from

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

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

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

The syntax of this MATLAB function is as follows:

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

% nutation in longitude and obliquity

% IAU 1980 version

% input

%  jdate = julian date

% output

%  dpsi = nutation in longitude (arc seconds)
%  deps = nutation in obliquity (arc seconds)
```

18. Precession Transformation

This MATLAB function (`precess.m`) precesses equatorial rectangular coordinates from one epoch to another. The coordinates are referred to the mean 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 George 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 as follows:

```
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
```

19. Kepler's Equation

This section describes three MATLAB functions that can be used to solve the *elliptic* form of Kepler's equation given by

$$M = E - e\sin E$$

where M is the mean anomaly, E is the eccentric anomaly and e is the orbital eccentricity. The two algorithms based on Professor Danby's work can also solve the hyperbolic form of Kepler's equation.

kepler1.m – Danby's method

This section describes a numerical solution devised by Professor J.M.A. Danby at North Carolina State University. Additional information about this algorithm can be found in "The Solution of Kepler's Equation", *Celestial Mechanics*, **31** (1983) 95-107, 317-328 and **40** (1987) 303-312.

The initial guess for Danby's method is

$$E_0 = M + 0.85\,\text{sign}\left(\sin M\right)e$$

The fundamental equation we want to solve is

$$f\left(E\right) = E - e\sin E - M = 0$$

which has the first three derivatives given by

$$\begin{aligned} f'\left(E\right) &= 1 - e\cos E \\ f''\left(E\right) &= e\sin E \\ f'''\left(E\right) &= e\cos E \end{aligned}$$

The iteration for an updated eccentric anomaly based on a current value E_n is given by the next four equations:

$$\begin{aligned} \Delta\left(E_n\right) &= -\frac{f}{f'} \\ \Delta^*\left(E_n\right) &= -\frac{f}{f' + \frac{1}{2}\Delta f''} \\ \Delta_n\left(E_n\right) &= -\frac{f}{f' + \frac{1}{2}\Delta f'' + \frac{1}{6}\Delta^{*2} f'''} \\ E_{n+1} &= E_n + \Delta_n \end{aligned}$$

This algorithm provides *quartic* convergence of Kepler's equation. This process is repeated until the following convergence test involving the fundamental equation is satisfied:

$$\left| f\left(E\right) \right| \le \varepsilon$$

where ε is the convergence tolerance. This tolerance is hardwired in the software to ε =1.0e-10. Finally, the true anomaly can be calculated with the following two equations

$$\begin{aligned} \sin\nu &= \sqrt{1-e^2}\,\sin E \\ \cos\nu &= \cos E - e \end{aligned}$$

and the four quadrant inverse tangent given by

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

If the orbit is hyperbolic the initial guess is

$$H_0 = \log\left(\frac{2M}{e} + 1.8\right)$$

where H_0 is the hyperbolic anomaly.

The fundamental equation and first three derivatives for this case are as follows:

$$\begin{aligned} f\left(H\right) &= e\sinh H - H - M \\ f'\left(H\right) &= e\cosh H - 1 \\ f''\left(H\right) &= e\sinh H \\ f'''\left(H\right) &= e\cosh H \end{aligned}$$

Otherwise, the iteration loop which calculates Δ, Δ^*, and so forth is the same. The true anomaly for hyperbolic orbits is determined with this next set of equations:

$$\begin{aligned} \sin\nu &= \sqrt{e^2-1}\,\sinh H \\ \cos\nu &= e - \cosh H \end{aligned}$$

The true anomaly is then determined from a four quadrant inverse tangent evaluation of these two equations.

The syntax of this MATLAB function is

```
function [eanom, tanom] = kepler1 (manom, ecc)

% solve Kepler's equation for circular,
% elliptic and hyperbolic orbits

% Danby's method

% input

%  manom = mean anomaly (radians)
%  ecc   = orbital eccentricity (non-dimensional)

% output

%  eanom = eccentric anomaly (radians)
%  tanom = true anomaly (radians)
```

kepler2.m – Danby's method with Mikkola's initial guess

This function solves the elliptic and hyperbolic form of Kepler's equation using Danby's method and Mikkola's initial guess. This method uses a cubic approximation of Kepler's equation for the initial guess. Additional information about this initial guess can be found in "A Cubic Approximation For Kepler's Equation", *Celestial Mechanics*, **40** (1987) 329-334.

The elliptic orbit initial guess for this method is given by the expression

$$E_0 = M + e\left(3s - 4s^3\right)$$

where

$$s = s_1 + ds$$

$$s_1 = z - \frac{\alpha}{2}$$

$$ds = -\frac{0.078 s_1^5}{1+e}$$

$$z = \operatorname{sign}(\beta)\left(\left|\beta\right| + \sqrt{\alpha^2 + \beta^2}\right)^{1/3}$$

$$\alpha = \frac{1-e}{4e + \frac{1}{2}}$$

$$\beta = \frac{\frac{1}{2}M}{4e + \frac{1}{2}}$$

Updates to the eccentric anomaly are calculated using the same set of equations as those in Danby's method. For hyperbolic orbits this method uses the following for the correction to *s*

$$ds = \frac{0.071 s^5}{e\left(1 + 0.45 s^2\right)\left(1 + 4 s^2\right)}$$

and the following initial guess for the hyperbolic anomaly:

$$H_0 = 3\log\left(s + \sqrt{1 + s^2}\right)$$

The syntax for this MATLAB function is

```
function [eanom, tanom] = kepler2 (manom, ecc)

% solve Kepler's equation for circular,
% elliptic and hyperbolic orbits

% Danby's method with Mikkola's initial guess

% input

%  manom = mean anomaly (radians)
%  ecc   = orbital eccentricity (non-dimensional)

% output

%  eanom = eccentric anomaly (radians)
%  tanom = true anomaly (radians)
```

kepler3.m – Gooding's two iteration method

This MATLAB function solves the elliptic form of Kepler's equation using Gooding's two iteration method. This algorithm always performs two, and only two iterations when solving Kepler's equation. Additional information about this technique can be found in "Procedures For Solving Kepler's Equation", *Celestial Mechanics*, **38** (1986) 307-334.

The syntax of this MATLAB function is

```
function [eanom, tanom] = kepler3 (manom, ecc)

% solve Kepler's equation for elliptic orbits

% Gooding's two iteration method

% input

%  manom = mean anomaly (radians)
%  ecc   = orbital eccentricity (non-dimensional)

% output

%  eanom = eccentric anomaly (radians)
%  tanom = true anomaly (radians)
```

kepler4.m – parabolic and near-parabolic orbits

This MATLAB function solves Kepler’s equation for heliocentric parabolic and near-parabolic orbits. It is based on the numerical method described in Chapter 4 of *Astronomy on the Personal Computer* by Oliver Montenbruck and Thomas Pfleger. This algorithm uses a modified form of Barker’s equation and Stumpff functions to solve this problem.

The form of Kepler’s equation solved by this function is

$$E(t) - e\sin E(t) = \sqrt{\frac{\mu}{a^3}}\,(t - t_0)$$

where

$E =$	eccentric anomaly
$e =$	orbital eccentricity
$\mu =$	gravitational constant of the Sun
$a =$	semimajor axis
$t =$	time
$t_0 =$	time of perihelion passage

The relationship between semimajor axis *a* and perihelion radius *q* is as follows:

$$a = \frac{q}{1-e}$$

By introducing the variable

$$U = \sqrt{\frac{3ec_3(E)}{1-e}}\,E$$

Kepler’s equation is now given by

$$U + \frac{1}{3}U^3 = \sqrt{6ec_3(E)}\sqrt{\frac{\mu}{2q^3}}\,(t - t_0)$$

where $c_3(E) = (E - e\sin E)/E^3$. The `kepler4` function iteratively solves for *U*.

The heliocentric distance is determined from the expression

$$r = q\left(1 + \left[\frac{2c_2}{6c_3}\right]U^2\right)$$

The true anomaly is determined from the *x* and *y* components of the heliocentric position vector as follows:

$$\nu = \operatorname{atan}\left(y, x\right)$$

where

$$x = q\left(1 - \left[\frac{2c_2}{6ec_3}\right]U^2\right)$$

$$y = 2q\sqrt{\frac{1+e}{2e}\left[\frac{1}{6c_3}\right]}c_1U$$

The true anomaly can also be determined from

$$\tan\left(\frac{\nu}{2}\right) = \left(\sqrt{\frac{1+e}{3ec_3}}\frac{c_2}{c_1}\right)U$$

The *c* functions used in these equations are called Stumpff functions. They are named after the German astronomer Karl Stumpff and defined by the series

$$c_n\left(x\right) = \sum_{k=0}^{\infty}\frac{\left(-1\right)^k x^k}{\left(2k+n\right)!} \qquad k = 0,1,2,\ldots$$

For *x* real and $x \neq 0$, the first few terms are given by the following expressions:

$$c_0\left(x^2\right) = \cos x \qquad c_0\left(-x^2\right) = \cosh x$$

$$c_1\left(x^2\right) = \frac{\sin x}{x} \qquad c_1\left(-x^2\right) = \frac{\sinh x}{x}$$

$$c_2\left(x^2\right) = \frac{1-\cos x}{x} \qquad c_2\left(-x^2\right) = \frac{\cosh x - 1}{x}$$

The Stumpff functions also satisfy the recursion relationship defined by

$$xc_{k+2}\left(x\right) = \frac{1}{k!} - c_k\left(x\right) \qquad k = 0,1,2,\ldots$$

For $x = 0$,

$$c_n\left(x\right) = \frac{1}{n!}$$

It is most efficient to compute c_2 and c_3 by series and then compute c_0 and c_1 by recursion according to the following:

$$c_0(x) = 1 - xc_2$$
$$c_1(x) = 1 - xc_3$$

The following is the syntax for this MATLAB function.

```
function [r, tanom] = kepler4(t, q, ecc)

% Kepler's equation for heliocentric
% parabolic and near-parabolic orbits

% input

%  t   = time relative to perihelion passage (days)
%  q   = perihelion radius (AU)
%  ecc = orbital eccentricity (non-dimensional)

% output

%  r     = heliocentric distance (AU)
%  tanom = true anomaly (radians)
```

Please note that the time relative to perihelion passage is in days and the perihelion radius is in astronomical units.

20. Two-body Orbital Motion

The following three MATLAB functions can be used to quickly and efficiently propagate two body or "unperturbed" orbits. For programming flexibility the input and output arguments for each routine are identical.

twobody1.m – Goodyear's method

This function is a MATLAB implementation of Goodyear's two body propagation algorithm.

Goodyear's method uses a change of variables that allows one to use a single set of equations to predict elliptic, hyperbolic and parabolic motion. This change of variables is defined by

$$\dot{\psi} = 1/r$$

where *r* is the distance of the spacecraft or celestial body.

The relationship between this generalized anomaly ψ and the classical eccentric anomaly *E* and hyperbolic anomaly *F* is given by

$$\psi = \frac{E - E_0}{\sqrt{\mu / a}} = \frac{F - F_0}{\sqrt{-\mu / a}}$$

The equations used to calculate the final position vector $\mathbf{r}$ and the final velocity vector $\dot{\mathbf{r}}$ from the initial position and velocity vectors $\mathbf{r}_0$ and $\dot{\mathbf{r}}_0$ are as follows:

$$r_0 = \sqrt{\mathbf{r}_0 \bullet \mathbf{r}_0}$$
$$\sigma_0 = \mathbf{r}_0 \bullet \dot{\mathbf{r}}_0$$
$$\alpha = \mathbf{r}_0 \bullet \dot{\mathbf{r}}_0 - 2\frac{\mu}{r_0}$$

The generalized eccentric anomaly ψ and the four transcendental functions given by

$$s_0 = 1 + \frac{\alpha\psi^2}{2!} + \frac{\alpha^2\psi^4}{4!} + \frac{\alpha^3\psi^6}{6!} + \dots$$
$$s_1 = \psi + \frac{\alpha\psi^3}{3!} + \frac{\alpha^2\psi^5}{5!} + \frac{\alpha^3\psi^7}{7!} + \dots$$
$$s_2 = \frac{\psi^2}{2!} + \frac{\alpha\psi^4}{4!} + \frac{\alpha^2\psi^6}{6!} + \frac{\alpha^3\psi^8}{8!} + \dots$$
$$s_3 = \frac{\psi^3}{3!} + \frac{\alpha\psi^5}{5!} + \frac{\alpha^2\psi^7}{7!} + \frac{\alpha^3\psi^9}{9!} + \dots$$

can be obtained by solving the next equation

$$t = t_0 + r_0 s_1 + \sigma_0 s_2 + \mu s_3$$

for the value of ψ which satisfies all these equations.

The position magnitude at the final time is given by

$$r = r_0 s_0 + \sigma_0 s_2 + \mu s_3$$

The *f* and *g* functions and their derivatives are calculated with the four equations given by

$$\begin{aligned} f &= 1 - \mu \frac{s_2}{r_0} \\ \dot{f} &= -\mu \frac{s_1}{r r_0} \\ g &= \left(t - t_0\right) - \mu s_3 \\ \dot{g} &= 1 - \mu \frac{s_2}{r} \end{aligned}$$

Finally, the position and velocity vectors at the final time *t* are computed from the *f* and *g* functions with the next two equations:

$$\begin{aligned} \mathbf{r}\left(t\right) &= f\,\mathbf{r}_0 + g\dot{\mathbf{r}}_0 \\ \dot{\mathbf{r}}\left(t\right) &= \dot{f}\,\mathbf{r}_0 + \dot{g}\dot{\mathbf{r}}_0 \end{aligned}$$

The two-body gravity acceleration vector at the final time is given by

$$\ddot{\mathbf{r}}\left(t\right) = -\mu \frac{\mathbf{r}}{\left|\mathbf{r}\right|^3}$$

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
tbcoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function.

The syntax of this MATLAB function is

```
function [rf, vf] = twobody1 (tau, ri, vi)
```

```
% solve the two body initial value problem

% Goodyear's method

% input

%  tau = propagation time interval (seconds)
%  ri  = initial eci position vector (kilometers)
%  vi  = initial eci velocity vector (km/sec)

% output

%  rf = final eci position vector (kilometers)
%  vf = final eci velocity vector (km/sec)
```

twobody2.m – Sheppard's method

This function is a MATLAB implementation of Stanley Shepperd's two body propagation algorithm. The derivation of this algorithm is described in "Universal Keplerian State Transition Matrix", *Celestial Mechanics*, **35** (1985) 129-144. This algorithm uses a combination of universal variables and continued fractions to propagate two-body orbits.

The syntax of this MATLAB function is

```
function [rf, vf] = twobody2 (tau, ri, vi)

% solution of the two body initial value problem

% Shepperd's method

% input

%  tau = propagation time interval (seconds)
%  ri  = initial eci position vector (kilometers)
%  vi  = initial eci velocity vector (km/sec)

% output

%  rf = final eci position vector (kilometers)
%  vf = final eci velocity vector (km/sec)
```

twobody3.m – Stumpff/Danby method

This function is a MATLAB implementation of the Stumpff/Danby two body propagation algorithm. It is based on a universal variable technique and Stumpff functions. This function is valid for elliptic and hyperbolic orbits. The derivation of this algorithm can be found in Professor Danby's classic text *Fundamentals of Celestial Mechanics*.

The algorithm begins by computing an initial guess for *s* according to

$$x = \alpha s^2$$

where

$$\mathbf{r}_0 = \text{ position vector at initial time } t_0$$
$$\mathbf{v}_0 = \text{ velocity vector at initial time } t_0$$
$$r_0 = |\mathbf{r}_0| = \sqrt{r_{0_x}^2 + r_{0_y}^2 + r_{0_z}^2}$$
$$v_0 = |\mathbf{v}_0| = \sqrt{v_{0_x}^2 + v_{0_y}^2 + v_{0_z}^2}$$
$$\alpha = \frac{2\mu}{r_0} - v_0^2$$

This algorithm uses a universal variable form of Kepler's equation defined by

$$f(s) = r_0 s c_1\left(\alpha s^2\right) + r_0 \dot{r}_0 s^2 c_2\left(\alpha s^2\right) + \mu s^3 c_3\left(\alpha s^2\right) - \left(t - t_0\right) = 0$$

The first three derivatives of this form of Kepler's equation are given by

$$f'(s) = r_0 c_0\left(\alpha s^2\right) + r_0 \dot{r}_0 s c_1\left(\alpha s^2\right) + \mu s^2 c_2\left(\alpha s^2\right)$$
$$f''(s) = \left(-r_0\alpha + \mu\right) s c_1\left(\alpha s^2\right) + r_0 \dot{r}_0 c_0\left(\alpha s^2\right)$$
$$f'''(s) = \left(-r_0\alpha + \mu\right) c_0\left(\alpha s^2\right) - r_0 \dot{r}_0 \alpha s c_1\left(\alpha s^2\right)$$

The c functions used in these equations are called Stumpff functions. They are named after the German astronomer Karl Stumpff and are defined by the series

$$c_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{(2k+n)!} \qquad k = 0,1,2,\ldots$$

for x real and $x \neq 0$

$$c_0\left(x^2\right) = \cos x \qquad c_0\left(-x^2\right) = \cosh x$$
$$c_1\left(x^2\right) = \frac{\sin x}{x} \qquad c_1\left(-x^2\right) = \frac{\sinh x}{x}$$
$$c_2\left(x^2\right) = \frac{1-\cos x}{x} \qquad c_2\left(-x^2\right) = \frac{\cosh x - 1}{x}$$

The Stumpff functions also satisfy the recursion relationship defined by

$$x c_{k+2}(x) = \frac{1}{k!} - c_k(x) \qquad k = 0,1,2,\ldots$$

For $x = 0$,

$$c_n\left(x\right)=\frac{1}{n!}$$

It is most efficient to compute c_2 and c_3 by series, and then compute c_0 and c_1 by recursion according to the following:

$$c_0\left(x\right)=1-xc_2$$
$$c_1\left(x\right)=1-xc_3$$

The equations for the *f* and *g* functions and derivatives are as follows:

$$f=1-\left(\frac{\mu}{r_0}\right)s^2c_2\left(\alpha s^2\right)$$
$$g=t-t_0-\mu s^3c_3\left(\alpha s^2\right)$$
$$\dot{f}=-\left(\frac{\mu}{rr_0}\right)sc_1\left(\alpha s^2\right)$$
$$\dot{g}=1-\left(\frac{\mu}{r}\right)s^2c_2\left(\alpha s^2\right)$$

Finally, the position and velocity vectors at the final time *t* are given by

$$\mathbf{r}\left(t\right)=f\,\mathbf{r}_0+g\dot{\mathbf{r}}_0$$
$$\dot{\mathbf{r}}\left(t\right)=\dot{f}\,\mathbf{r}_0+\dot{g}\dot{\mathbf{r}}_0$$

The syntax of this MATLAB function is

```
function [rf, vf] = twobody3 (tau, ri, vi)

% solve the two body initial value problem

% Danby/Stumpff method

% input

%  tau = propagation time interval (seconds)
%  ri  = initial eci position vector (kilometers)
%  vi  = initial eci velocity vector (km/sec)

% output

%  rf = final eci position vector (kilometers)
%  vf = final eci velocity vector (km/sec)
```

21. Time and Date Functions

This section describes several MATLAB functions that can be used for time and date conversions and manipulations.

julian.m – calendar date to Julian date

This MATLAB function converts a calendar date to its corresponding Julian date. Be sure to pass this routine all digits of the calendar year.

The syntax of this MATLAB function is

```
function jdate = julian (month, day, year)

% Julian date subroutine

% Input

%  month = calendar month [1 - 12]
%  day   = calendar day [1 - 31]
%  year  = calendar year [yyyy]

% Output

%  jdate = Julian date
```

gdate.m – julian date to calendar date

This MATLAB function converts a Julian date to its corresponding calendar date.

The syntax of this MATLAB function is

```
function [month, day, year] = gdate (jdate)

% convert Julian date to Gregorian (calendar) date

% input

%  jdate = Julian date

% output

%  month = calendar month [1 - 12]
%  day   = calendar day [1 - 31]
%  year  = calendar year [yyyy]
```

jd2str.m – julian date to calendar date and universal time strings

This function converts a Julian date to its equivalent calendar date and universal time strings.

The syntax of this MATLAB function is

```
function [cdstr, utstr] = jd2str(jdate)
```

```
% convert Julian date to string equivalent
% calendar date and universal time

% input

%  jdate = Julian date

% output

%  cdstr = calendar date string
%  utstr = universal time string
```

utc2tdt.m – convert UTC Julian date to TDT Julian date

This function converts a Julian date on the coordinated universal time (UTC) time scale to a Julian date on the terrestrial dynamic time (TDT) scale. It is valid between January 1, 1940 and the near future.

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
itime = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function.

The syntax of this MATLAB function is

```
function jdtdt = utc2tdt(jdutc)

% convert utc julian date to tdt julian date

% input

%  jdutc = universal time julian date

% output

%  jdtdt = terrestrial dynamic time julian date
```

tdt2tdb.m – convert TDT Julian date to TDB Julian date

This function converts a Julian date on the terrestrial dynamic time (TDT) also called terrestrial time TT) time scale to a Julian date on the barycentric dynamic time (TDB) scale. TDB is the fundamental time argument for the Jet Propulsion Laboratory (JPL) and Bureau of Longitudes (BDL) ephemerides. The difference between these two time scales can be as much as 1.6 milliseconds. This difference is periodic with an average of zero.

The syntax of this MATLAB function is

```
function jdtdb = tdt2tdb(jdtdt)

% convert terrestrial dynamic time julian
% date to barycentric dynamic time julian date

% input

%  jdtdt = tdt julian date

% output

%  jdtdb = tdb julian date
```

hrs2hms.m – convert hours function

This function can be used to convert a floating point argument in hours into its equivalent degrees, minutes, seconds, and string representation.

The syntax of this MATLAB function is

```
function [h, m, s, hmsstr] = hrs2hms (hrs)

% convert decimal hours to hours,
% minutes, seconds and equivalent string

% input

%  dd = angle in hours

% output

%  h      = integer hours
%  m      = integer minutes
%  s      = seconds
%  hmsstr = string equivalent of input
```

22. Numerical Methods and Utility Routines

This section describes several fundamental MATLAB functions that can be used to create custom applications.

TRIGONOMETRY AND VECTOR ROUTINES

atan3.m – four quadrant inverse tangent

This function calculates the four quadrant inverse tangent.

The syntax of this MATLAB function is

```
function y = atan3 (a, b)

% four quadrant inverse tangent

% input

%  a = sine of angle
%  b = cosine of angle

% output

%  y = angle (radians; 0 =< c <= 2 * pi)
```

modulo.m – modulo 2 pi

This function will modulo an angle (in radians) between 0 and 2 π.

The syntax of this MATLAB function is

```
function y = modulo (x)

% modulo 2 pi function

% input

%  x = argument (radians)
% output

%  y = x modulo 2 pi
```

uvector.m – unit vector

This function calculates a unit vector from a user provided three component vector.

The syntax of this MATLAB function is

```
function vhat = uvector(x)

% unit vector
% input

%  x = 3-component input vector

% output

%  vhat = 3-component unit vector
```

DIFFERENTIAL EQUATIONS

rkf45.m – Runge-Kutta-Fehlberg 4(5) method for first-order systems

This function is a MATLAB implementation of the 4(5) version of the Runge-Kutta-Fehlberg algorithm. It can be used to solve systems of first order, nonlinear vector differential equations of the form $\dot{\mathbf{y}} = f(\mathbf{y}, t)$.

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
rkcoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function. After the first call, the function will set this value to `0`.

The syntax of this MATLAB function is

```
function xout = rkf45 (deq, neq, ti, tf, h, tetol, y)

% solution of first order system of differential equations

% Runge-Kutta-Fehlberg 4(5)

% fourth-order solution with fifth-order error control

% input

%  neq   = number of equations in system
%  ti    = initial simulation time
%  tf    = final simulation time
%  h     = guess for step size
%  tetol = truncation error tolerance
%  y     = initial integration vector

% output

%  xout = final integration vector
```

rkf78.m – Runge-Kutta-Fehlberg 7(8) method for first-order systems

This function is a MATLAB implementation of the 7(8) version of the Runge-Kutta-Fehlberg algorithm. It can be used to solve systems of first order, nonlinear vector differential equations of the form $\dot{\mathbf{y}} = f(\mathbf{y}, t)$.

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
rkcoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function. After the first call, the function will set this value to `0`.

Near the end of this function is the following code:

```
if (xerr > 1)
      % reject current step
      ti = twrk;
      x = xwrk;
   else
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      % accept current step              %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      % perform graphics, additional     %
      % calculations, etc. at this point %
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   end
```

When the processing reaches the `else` statement, a successful integration step has been completed and additional processing such as graphics or other calculations can be performed here. For example, the following code performs the rectification required for Encke's method.

```
if (xerr > 1)
      % reject current step

      ti = twrk;
      x = xwrk;

      % compute new step size

      dt = 0.7 * dt * (1 / xerr) ^ (1 / 8);
   else
      % rectify and re-initialize

      ytbf = twobody4(dt, ytbi);

      for i = 1:1:6
          ytrue(i) = ytbf(i) + x(i);

          ytbi(i) = ytrue(i);

          x(i) = 0;
      end

      tsaved = ti;
   end
```

The following is the syntax for this MATLAB function.

```
function xout = rkf78 (deq, neq, ti, tf, h, tetol, x)
```

```
% solve first order system of differential equations

% Runge-Kutta-Fehlberg 7(8) method

% input

%  deq   = name of function which defines the
%          system of differential equations
%  neq   = number of differential equations
%  ti    = initial simulation time
%  tf    = final simulation time
%  h     = initial guess for integration step size
%  tetol = truncation error tolerance (non-dimensional)
%  x     = integration vector at time = ti

% output

%  xout  = integration vector at time = tf
```

nym4.m – Nystrom fourth-order method for second-order systems

This function is a MATLAB implementation of a fourth-order Nystrom algorithm. It can be used to solve systems of second order differential equations of the form $\ddot{\mathbf{x}} = f\left(\mathbf{x}, \dot{\mathbf{x}}, t\right)$.

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
nycoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script that calls this function.

The syntax of this MATLAB function is

```
function [rf, vf] = nym4(deq, n, tp, dt, rs, vs)

% solve a system of second order differential equations

% fourth order Nystrom method (fixed step size)

% input

%  deq = function name of systems of
%        differential equations
%  n   = number of equations in user-defined
%        system of differential equations
%  tp  = current simulation time
%  dt  = integration step size
%  rs  = position vector at initial time
%  vs  = velocity vector at initial time

% output

%  rf = position vector at time = tp + dt
%  vf = velocity vector at time = tp + dt
```

ceqm1.m – first-order heliocentric equations of orbital motion

This MATLAB function evaluates the first-order form of the heliocentric equations of orbital motion. It includes the point-mass gravity perturbations due to Venus, Earth, Mars, Jupiter and Saturn. The fundamental coordinate system used in this algorithm is the true-of-date ecliptic and equinox.

The syntax of this MATLAB function is

```
function yddot = ceqm1 (time, y)

% first order form of the heliocentric
% equations of motion

% includes point mass gravity
% perturbations of Venus, Earth,
% Mars, Jupiter and Saturn

% input

%  time = current simulation time
%  y    = current state vector

% output

%  yddot = equations of motion
```

The second-order heliocentric equations of motion of a satellite or celestial body subject to the *point-mass* gravitational attraction of the Sun and planets are given by

$$\ddot{\mathbf{r}} = \frac{d^2\mathbf{r}}{dt^2}\left(\mathbf{r},t\right) = -\mu_s \frac{\mathbf{r}_{s-b}}{\left|\mathbf{r}_{s-b}\right|^3} - \sum_{i=1}^{9} \mu_{p_i} \left(\frac{\mathbf{r}_{(p-b)_i}}{\left|\mathbf{r}_{(p-b)_i}\right|^3} + \frac{\mathbf{r}_{p_i}}{\left|\mathbf{r}_{p_i}\right|^3} \right)$$

where

$\mu_s =$ gravitational constant of the Sun

$\mu_{p_i} =$ gravitational constant of planet i

$\mathbf{r}_p =$ position vector from the Sun to planet

$\mathbf{r}_{s-b} =$ position vector from the Sun to the body

$\mathbf{r}_{p-b} =$ position vector from the planet to the body

These position vectors are related according to

$$\mathbf{r}_{s-b} = \mathbf{r}_p + \mathbf{r}_{p-b}$$

meeeqm.m – modified equinoctial equations of motion

This MATLAB function evaluates the modified equinoctial orbital elements form of the equations of heliocentric orbital motion.

The syntax for this MATLAB function is

```
function ydot = meeeqm(t, y)

% modified equinoctial equations of motion

% input

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

% output

%  ydot = first time derivatives of
%         modified equinoctial elements
```

The perturbation vector used in these calculations is computed in a MATLAB function called `ceqm2.m`. The syntax of this function is as follows:

```
function ywrk = ceqm2 (t, r, v)

% eci perturbation vector

% support function for cowell4.m

% includes perturbations due to:

%   non-spherical earth gravity
%   aerodynamic drag (us 76 model)
%   solar radiation pressure
%   sun and moon

% input

%  t = current simulation time
%  r = current eci position vector
%  v = current eci velocity vector

% output

%  ywrk = eci perturbation vector
```

The equations of orbital motion expressed in terms of modified equinoctial orbital elements are as follows:

$$\dot{p} = \frac{dp}{dt} = \frac{2p}{w}\sqrt{\frac{p}{\mu}}\,\Delta_t$$

$$\dot{f} = \frac{df}{dt} = \sqrt{\frac{p}{\mu}}\left[\Delta_r \sin L + \left[(w+1)\cos L + f\right]\frac{\Delta_t}{w} - (h\sin L - k\cos L)\frac{g\Delta_n}{w}\right]$$

$$\dot{g} = \frac{dg}{dt} = \sqrt{\frac{p}{\mu}}\left[-\Delta_r \cos L + \left[(w+1)\sin L + g\right]\frac{\Delta_t}{w} + (h\sin L - k\cos L)\frac{g\Delta_n}{w}\right]$$

$$\dot{h} = \frac{dh}{dt} = \sqrt{\frac{p}{\mu}}\,\frac{s^2\Delta_n}{2w}\cos L$$

$$\dot{k} = \frac{dk}{dt} = \sqrt{\frac{p}{\mu}}\,\frac{s^2\Delta_n}{2w}\sin L$$

$$\dot{L} = \frac{dL}{dt} = \sqrt{\mu p}\left(\frac{w}{p}\right)^2 + \frac{1}{w}\sqrt{\frac{p}{\mu}}(h\sin L - k\cos L)\Delta_n$$

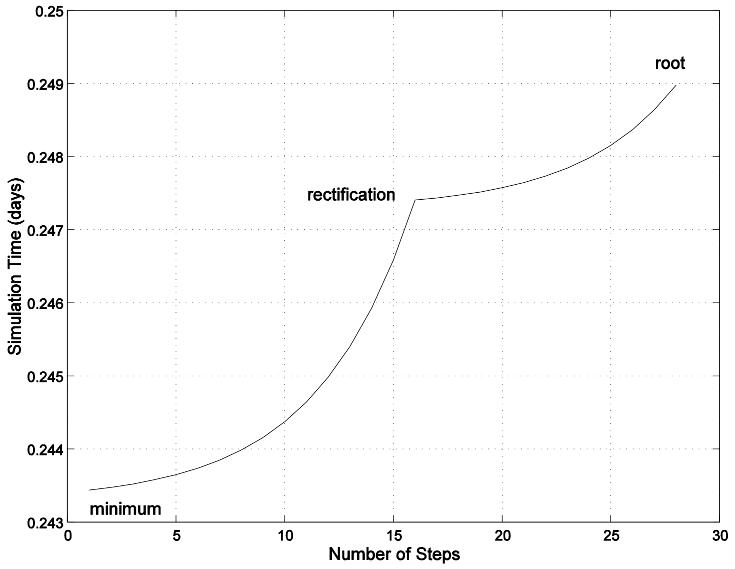
where $\Delta_r, \Delta_t, \Delta_n$ are external perturbations in the radial, tangential and orbit normal directions, respectively.

ROOT-FINDING AND OPTIMIZATION

broot.m – bracket a single root of a nonlinear function

This function can be used to bracket a single root of a single nonlinear equation of the form $y = f(x)$. It uses a combination of *geometric acceleration* and *rectification* to bracket roots. The basic idea is to find the endpoints of an interval x_1 and x_2 such that $f(x_1)\,f(x_2) < 0$. The user must supply this function with an initial guess for x_1 and x_2. Typically x_1 is the time of an objective function minimum or maximum and x_2 is a time 10 seconds after (forward root) or 10 seconds before (backward root) the value of x_1

The following diagram illustrates this process. The acceleration factor for this example is 0.25 and the rectification interval is 300 seconds. The acceleration factor increases the size of each step geometrically and the rectification interval monitors the length of the current bracketing interval. When necessary the rectification logic reinitializes the search and prevents the interval from becoming too large and skipping one or both ends of the bracketing interval completely. The process starts at the bottom left, labeled *minimum*, which is the time of the function minimum for this example. The *x*-axis is the number of steps taken in the search for the bracketing interval and the *y*-axis is the elapsed simulation time. The size of each step increases geometrically until the length of the current step reaches the value of the rectification interval. At that point, labeled *rectification* on the plot, the step size is reset and the process begins again. Eventually, the process will bracket the root, labeled *root* on this plot, and the function will return the endpoints x_1 and x_2 of this bracketing interval.



The syntax of this MATLAB function is

```
function [x1out, x2out] = broot (f, x1in, x2in, factor, dxmax)

% bracket a single root of a nonlinear equation

% input

%  f      = objective function coded as y = f(x)
%  x1in   = initial guess for first bracketing x value
%  x2in   = initial guess for second bracketing x value
%  factor = acceleration factor (non-dimensional)
%  dxmax  = rectification interval

% output

%  x1out = final value for first bracketing x value
%  x2out = final value for second bracketing x value
```

brent.m – solve for a single root of a nonlinear function

This function uses Brent’s method to find a single root of a single nonlinear equation of the form $y = f(x)$. Derivatives are not required for this algorithm. However, the user should ensure that a root is bracketed $\left(f(x_1) f(x_2) < 0 \right)$ before calling this routine.

The syntax of this MATLAB function is

```
function [xroot, froot] = brent (f, x1, x2, rtol)

% solve for a single real root of a nonlinear equation

% Brent's method

% input

%  f    = objective function coded as y = f(x)
%  x1   = lower bound of search interval
%  x2   = upper bound of search interval
%  rtol = algorithm convergence criterion

% output

%  xroot  = real root of f(x) = 0
%  froot  = function value at f(x) = 0
```

minima.m – one-dimensional minimization

This function uses Brent’s method to find a single minimum or maximum of a single user-defined nonlinear “objective” function.

The syntax of this MATLAB function is

```
function [xmin, fmin] = minima (f, a, b, tolm)

% one-dimensional minimization subroutine

% Brent's method

% input

%  f    = objective function coded as y = f(x)
%  a    = initial x search value
%  b    = final x search value
%  tolm = convergence criterion

% output

%  xmin = minimum x value
%  fmin = minimum function value
```

oevent1.m – find minimization/root finding orbital event

This MATLAB function is a combined one-dimensional minimization and root-finding algorithm. It first finds minima or maxima within a user-defined search interval, and then calculates the corresponding “reverse” and “forward” root.

The syntax of this MATLAB function is

```
function oevent1 (objfunc, prtfunc, ti, tf, dt, dtsml)

% predict minimization/root-finding orbital events
% input
%  objfunc = objective function
%  prtfunc = display results function
%  ti      = initial simulation time
%  tf      = final simulation time
%  dt      = step size used for bounding minima
%  dtsml   = small step size used to determine whether
%            the function is increasing or decreasing
```

ATMOSPHERIC REFRACTION

refract.m – refraction correction function

This MATLAB function corrects a topocentric elevation angle for atmospheric refraction. The correction to be added to the "true" or 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)
```

refract1.m – NOVAS refraction correction function

This MATLAB function determines the effect of atmospheric refraction using the algorithm included in the NOVAS software suite from the Naval Observatory.

The syntax of this MATLAB function is

```
function refr = refract1 (height, zdobs)

% this subroutine computes atmospheric refraction in zenith
% distance. this version computes approximate refraction for
% optical wavelengths. it can be used for planning observations
% or telescope pointing, but should not be used for the reduction
% of precise observations. basic algorithm is described in the
% explanatory supplement to the astronomical almanac, p. 144,
% and is an adaptation of a formula in bennett (1982), journal
% of navigation (royal institute) 35, 255-259.

%      height = height of observer in meters (in)
%       zdobs  = observed zenith distance in degrees (in)
%       refr   = atmospheric refraction in degrees (out)
```

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)
```

UTILITY ROUTINES

oeprint1.m – formatted classical orbital elements screen display

This function prints a formatted display of the six classical orbital elements with the orbital period displayed in days. There is also a version of this function called `oeprint2.m` that prints the semimajor axis in astronomical units.

The syntax of this MATLAB function is

```
function oeprint1(oev)

% print six classical orbital elements
% (orbital period in days)

% 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 periapsis (radians)
%            (0 <= argument of periapsis <= 2 pi)
%  oev(5)  = right ascension of ascending node (radians)
%            (0 <= raan <= 2 pi)
%  oev(6)  = true anomaly (radians)
%            (0 <= true 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 = position vector (kilometers)
%  v = velocity vector (kilometers/second)
```

deg2dms.m – convert degrees function

This function can be used to convert a floating point argument in degrees into its equivalent degrees, minutes, seconds, and string representation. For example, the command

```
    [d,m,s,degstr]=deg2dms(22.12345)
```

produces the following screen output:

```
    +22d 07m 24.42s
```

The syntax of this MATLAB function is

```
function [d, m, s, dmsstr] = deg2dms (dd)

% convert decimal degrees to degrees,
% minutes, seconds and equivalent string

% input

%  dd = angle in decimal degrees

% output

%  d      = integer degrees
%  m      = integer minutes
%  s      = seconds
%  dmsstr = string equivalent of input
```

INTERACTIVE REQUEST OF PROGRAM INPUTS

getdate.m – request calendar date

This function interactively requests the calendar date. Please note that all digits of the calendar year are required.

The syntax of this MATLAB function is

```
function [m, d, y] = getdate

% interactive request and input of calendar date

% output

%  m = calendar month
%  d = calendar day
%  y = calendar year
```

The following is the screen display for this function. Please note the valid range of inputs.

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

The calendar date should be input with individual elements separated by commas.

getobs.m – request observer coordinates

This function interactively requests the geodetic coordinates of a ground site. Please note that geodetic latitude and longitude are returned in the units of radians.

The syntax of this MATLAB function is

```
function [obslat, obslong, obsalt] = getobs

% interactive request of ground site coordinates

% output

%  obslat  = geographic latitude (radians)
%  obslong = geographic longitude (radians)
%  obsalt  = geodetic altitude (kilometers)
```

The following is a typical user interaction with this function. Please note the sign conventions and range of valid inputs. North latitudes are positive and south latitudes are negative. East longitude is positive and west longitude is negative.

```
please input the geographic latitude of the ground site
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 40,35,10

please input the geographic longitude of the ground site
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -105,30,45

please input the altitude of the ground site (meters)
(positive above sea level, negative below sea level)
? 100
```

getoe.m – request classical orbital elements

This function interactively requests six classical orbital elements. Please note that all angular elements are returned from this function in the units of radians. The user can control which orbital elements are requested by setting the corresponding value of the `ioev` vector. For example, the following statement will request all six orbital elements:

```
oev = getoe([1;1;1;1;1;1])
```

If the orbital eccentricity input by the user is 0, this function will bypass the prompt for argument of periapsis (if requested via `ioev`) and set its value to 0.

The syntax of this MATLAB function is

```
function oev = getoe(ioev)

% interactive request of classical orbital elements

% NOTE: all angular elements are returned in radians

% input

%  ioev = request array (1 = yes, 0 = no)

% output

%  oev(1) = semimajor
%  oev(2) = orbital eccentricity
%  oev(3) = orbital inclination
%  oev(4) = argument of perigee
%  oev(5) = right ascension of the ascending node
%  oev(6) = true anomaly
```

gettime.m – request universal time

This function interactively requests the universal time in hours, minutes and seconds.

The syntax of this MATLAB function is

```
function [uthr, utmin, utsec] = gettime

% interactive request and input of universal time

% output

%  uthr  = universal time (hours)
%  utmin = universal time (minutes)
%  utsec = universal time (seconds)
```

The following is the screen prompt displayed by this function. Please note the range of valid inputs. If invalid data is input the function will redisplay this request.

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

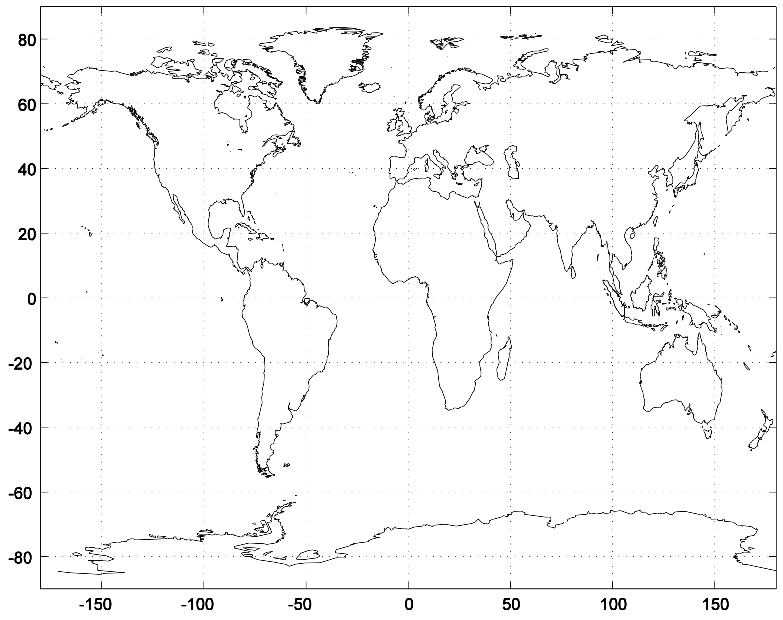
RECTANGULAR AND ORTHOGRAPHIC GRAPHICS

demomwdb.m – MicroWorld Database map files

This MATLAB application demonstrates how to read and plot the MicroWorld Database binary map files. The *Celestial Computing with MATLAB* software package includes versions of these maps for IBM-PC compatible computers. The map features that are plotted by this utility are defined in the code as follows:

```
% define plot features
% (set to 1 to activate);

allfeatures = 0;
coastlines = 1;
countries = 0;
states = 0;
islands = 1;
lakes = 0;
rivers = 0;
```

The following is a typical plot created with this script.



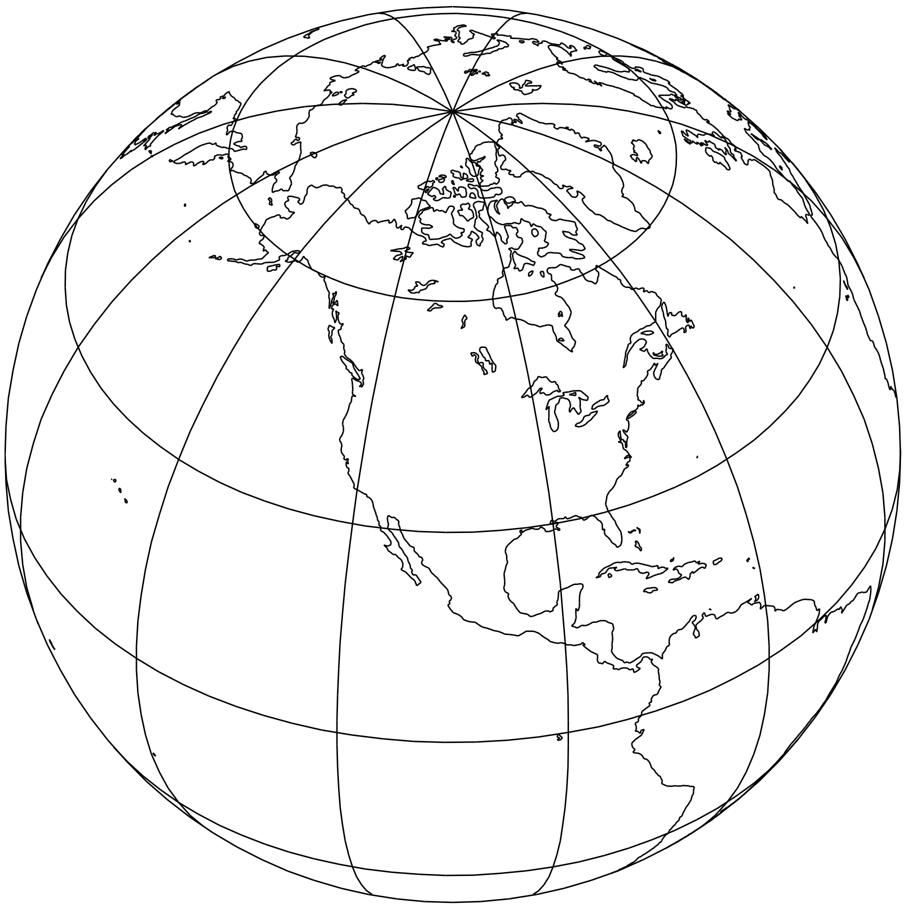
pltortho.m – plot orthographic view of the Earth

This MATLAB function will plot an orthographic view of the Earth. The map coordinates are contained in a file called `xyzmap.dat`. The *Celestial Computing with MATLAB* CD ROM contains a script called `demoplot.m` that demonstrates how to interact with both the `pltortho` and `pltrect` mapping functions.

The syntax of this MATLAB function is

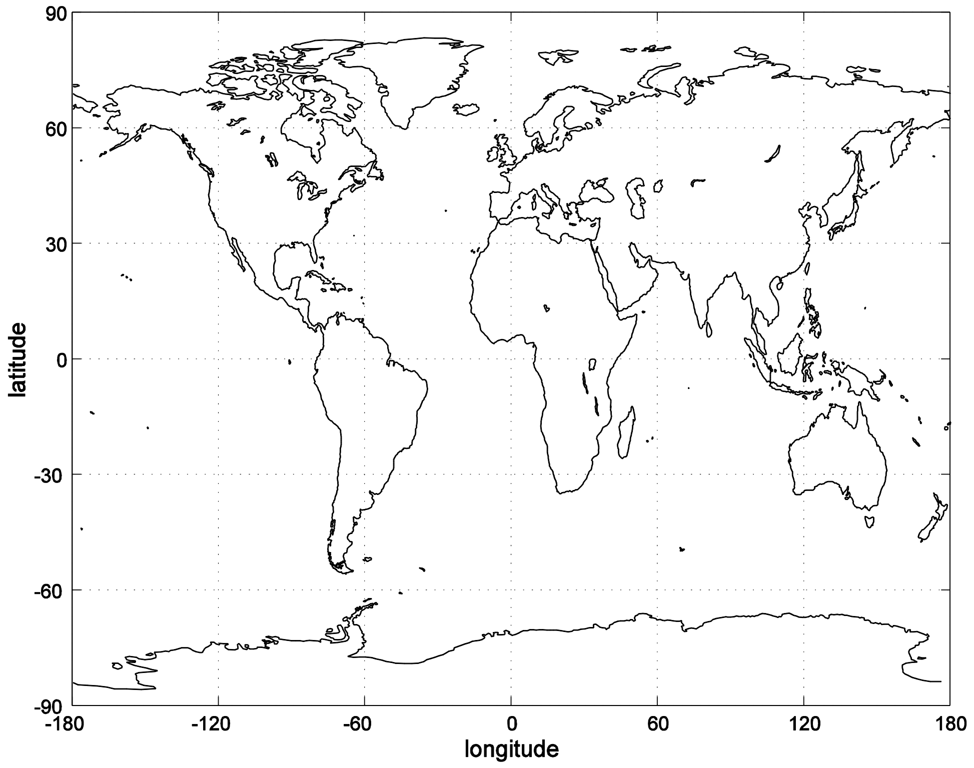
```
function pltortho(vplat, vplong, dlat, dlong)

% plot orthographic world map

% input

%  vplat  = viewpoint latitude (degrees)
%           (-90 <= vplat <= 90)
%  vplong = viewpoint longitude (degrees)
%           (-180 <= vplong <= 180)
%  dlat   = latitude grid spacing (degrees)
%           (0 <= dlat <= 90)
%  dlong  = longitude grid spacing (degrees)
%           (0 <= dlong <= 360)

% note

%  east longitudes are positive,
%  west longitudes are negative
```

The following is a typical plot created with this function. The viewpoint is 40 degrees north latitude and 105 degrees west longitude. The latitude and longitude grid spacing for this example was 30 degrees.



pltrect.m – plot rectangular map of the Earth

This function will plot the major features of the Earth on a rectangular map. The map coordinates are contained in a file called `gmapll.dat`. The syntax of this function is simply `function pltrect`.



23. Astronomical Coordinates

This section describes several MATLAB functions that can be used to calculate and convert different types of astronomical coordinates.

In the following discussions

$$\begin{aligned}
\alpha &= \text{ right ascension} \\
\delta &= \text{ declination} \\
\lambda &= \text{ ecliptic longitude} \\
\beta &= \text{ ecliptic latitude} \\
A &= \text{ topocentric azimuth} \\
h &= \text{ topocentric altitude} \\
H &= \text{ hour angle } = \theta - \alpha \\
\theta &= \text{ observer's local sidereal time} \\
\phi &= \text{ observer's geodetic latitude} \\
\varepsilon &= \text{ obliquity of the ecliptic}
\end{aligned}$$

Also, equations involving a two argument inverse tangent denote calculations that are performed with the four quadrant inverse tangent function `atan3.m`.

rec2pol.m – rectangular to polar coordinate conversion

The scalar distance can be determined from the three components of the rectangular position vector as follows:

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

The longitudinal polar coordinate is computed using a four quadrant inverse tangent function as follows:

$$\theta = \tan^{-1}\left(r_y, r_x\right)$$

The latitudinal polar coordinate is calculated using the expression given by

$$\sigma = \tan^{-1}\left(\frac{r_z}{\sqrt{r_x^2 + r_y^2}}\right)$$

The syntax of this MATLAB function is

```
function [rmag, lambda, phi] = rec2pol(r)

% rectangular to polar coordinate conversion
% input

%  r = rectangular position vector

% output

%  rmag   = vector scalar magnitude
%  lambda = longitudinal polar coordinate (radians)
%           (0 <= lambda <= 2*pi)
%  phi    = latitudinal polar coordinate (radians)
%           (-pi/2 <= phi <= pi/2)
```

pol2rec.m – polar to rectangular coordinate conversion

The three components of the rectangular position vector are determined from the scalar distance ρ, the longitudinal polar coordinate θ and the latitudinal polar coordinate σ as follows:

$$\mathbf{r} = \rho \begin{pmatrix} \cos\sigma\cos\theta \\ \sin\sigma\cos\theta \\ \sin\theta \end{pmatrix}$$

The syntax of this MATLAB function is

```
function r = pol2rec(rmag, lambda, phi)

% polar to rectangular coordinate conversion

% input

%  rmag   = vector scalar magnitude
%  lambda = longitudinal polar coordinate (radians)
%           (0 <= lambda <= 2*pi)
%  phi    = latitudinal polar coordinate (radians)
%           (-pi/2 <= phi <= pi/2)

% output

%  r = rectangular position vector
```

ecl2equ.m – ecliptic to equatorial coordinate conversion

The conversion from ecliptic to equatorial coordinates is given by the following two equations:

$$\alpha = \tan^{-1}\left(\sin\lambda\cos\varepsilon - \tan\beta\sin\varepsilon, \cos\lambda\right)$$

$$\delta = \sin^{-1}\left(\sin\beta\cos\varepsilon + \cos\beta\sin\varepsilon\sin\lambda\right)$$

The syntax of this MATLAB function is

```
function [rasc, decl] = ecl2equ(lat, lon, obliq)

% ecliptic to equatorial coordinate conversion
% input

%  lat   = ecliptic latitude (radians)
%  lon   = ecliptic longitude (radians)
%  obliq = obliquity of the ecliptic (radians)

% output

%  rasc = geocentric right ascension (radians)
%  decl = geocentric declination (radians)
```

equ2ecl.m – equatorial to ecliptic coordinate conversion

The following two equations convert equatorial coordinates to ecliptic coordinates:

$$\lambda = \tan^{-1}\left(\sin\alpha\cos\varepsilon + \tan\delta\sin\varepsilon, \cos\alpha\right)$$

$$\beta = \sin^{-1}\left(\sin\delta\cos\varepsilon - \cos\delta\sin\varepsilon\sin\alpha\right)$$

The syntax of this MATLAB function is

```
function [lat, long] = equ2ecl(decl, rasc, obliq)

% equatorial to ecliptic coordinate conversion

% input

%  rasc  = geocentric right ascension (radians)
%  decl  = geocentric declination (radians)
%  obliq = obliquity of the ecliptic (radians)

% output

%  lat = ecliptic latitude (radians)
%  lon = ecliptic longitude (radians)
```

equ2hor.m – equatorial to local horizontal coordinate conversion

The conversion from equatorial to local horizontal coordinates is given by the following two expressions:

$$A = \tan^{-1}\left(\sin H, \cos H\sin\phi - \tan\delta\cos\phi\right)$$

$$h = \sin^{-1}\left(\sin\phi\sin\delta + \cos\phi\cos\delta\cos H\right)$$

The syntax of this MATLAB function is

```
function [azimuth, elevation] = equ2hor(obslat, obslst, decl, rasc)

% equatorial to local horizontal coordinate conversion

% input
% obslat = observer's geodetic latitude (radians)
% obslst = observer's local sidereal time (radians)
% decl   = object's declination (radians)
% rasc   = object's right ascension (radians)

% output

% azimuth   = object's topocentric azimuth angle (radians)
% elevation = object's topocentric elevation angle (radians)

% notes

% (1) azimuth positive clockwise from north
% (2) elevation positive above the local horizontal
```

hor2equ.m – local horizontal to equatorial coordinate conversion

The conversion from local horizontal to equatorial coordinates is as follows:

$$H = \tan^{-1}\left(\sin A, \cos A \cos\phi + \tan h \cos\phi\right)$$

$$\delta = \sin^{-1}\left(\sin\phi \sin h - \cos\phi \cos h \cos A\right)$$

The syntax of this MATLAB function is

```
function [decl, rasc] = hor2equ(obslat, obslst, azimuth, elevation)

% local horizontal to equatorial coordinate conversion

% input

% azimuth   = object's topocentric azimuth angle (radians)
% elevation = object's topocentric elevation angle (radians)
% obslat    = observer's geodetic latitude (radians)
% obslst    = observer's local sidereal time (radians)

% output

% decl = object's declination (radians)
% rasc = object's right ascension (radians)

% notes

% (1) azimuth positive clockwise from north
% (2) elevation positive above the local horizontal
```

oepreces.m – transform angular orbital elements from one equinox to another

This MATLAB function can be used to transform angular orbital elements from one equinox to another. The orbital inclination i, argument of periapsis ω, and longitude of ascending node Ω are the only orbital elements that depend on the position of the equinox.

In the equations which follow, i_0, Ω_0, and ω_0 are the orbital inclination, longitude of ascending node and argument of periapsis with respect to the initial equinox. Orbital elements without a subscript are with respect to the final equinox. It is important to remember that these transformed orbital elements are still valid for the same epoch as the original orbital elements. In other words, this MATLAB function performs a coordinate transformation, not an epoch transformation.

The spherical trigonometry relationships among these elements are summarized below:

$$\cos i = \cos i_0 \cos \pi_A + \sin i_0 \sin \pi_A \cos\left(\Pi_A - \Omega_0\right)$$

$$\sin i \sin\left(p_A + \Pi_A - \Omega\right) = \sin i_0 \sin\left(\Pi_A - \Omega_0\right)$$

$$\sin i \cos\left(p_A + \Pi_A - \Omega\right) = -\sin \pi_A \cos i_0 + \cos \pi_A \sin i_0 \cos\left(\Pi_A - \Omega_0\right)$$

The correction to the argument of perapsis can be calculated from the following two expressions:

$$\sin \Delta\omega \sin i = \sin i_0 \cos \pi_A - \cos i_0 \sin \pi_A \cos\left(\Pi_A - \Omega_0\right)$$

$$\cos \Delta\omega \sin i = \sin i_0 \cos \pi_A - \cos i_0 \sin \pi_A \cos\left(\Pi_A - \Omega_0\right)$$

The argument of periapsis corrected for precession is as follows:

$$\omega = \omega_0 + \Delta\omega$$

In these expressions π_A, Π_A, and p_A are fundamental precessional quantities which are functions of time. They are described in “Precession Matrix Based on IAU (1976) System of Astronomical Constants”, *Astronomy and Astrophysics*, 73, 282-284 (1979) by J. H. Lieske.

This function will accept initial and final equinoxes in the form of Besselian (B1950 for example) and Julian years (J2000 for example) as well as Julian ephemeris dates (2451545 for example). These equinoxes should be passed to this function as strings.

The syntax of this MATLAB function is

```
function oev2 = oepreces (eqnx1, eqnx2, oev1)

% transform angular orbital elements
% from one equinox to another equinox

% input

% eqnx1 = initial equinox
% eqnx2 = final equinox
% oev1  = orbital elements referred to eqnx1
%         oev1(1) = orbital inclination
%         oev1(2) = longitude of ascending node
%         oev1(3) = argument of periapsis

% output
%  oev2 = orbital elements referred to eqnx2
%         oev2(1) = orbital inclination
%         oev2(2) = longitude of ascending node
%         oev2(3) = argument of periapsis
```

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.

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

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

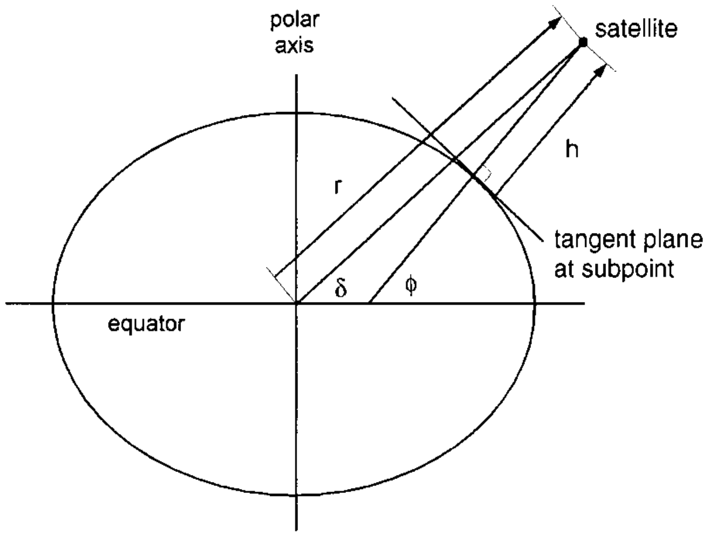
f = Earth flattening factor
r_{eq} = Earth equatorial radius
ϕ = geodetic latitude
h = geodetic altitude

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)
```

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

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

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

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

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

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

In these equations, ρ is the geocentric distance of the object, $\hat{h} = h / r_{eq}$ 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 exact 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 numerical method are as follows:

$$E = \frac{\left[b r_z - \left(a^2 - b^2 \right) \right]}{ar}$$
$$F = \frac{\left[b r_z + \left(a^2 - b^2 \right) \right]}{ar}$$
$$r = \sqrt{r_x^2 + r_y^2}$$

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

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

The calculations continue with

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

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

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

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

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

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

The geodetic latitude is determined from the expression

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

and the geodetic altitude is calculated from

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

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 = Earth flattening factor

r_{eq} = Earth equatorial radius

ϕ = geodetic latitude

λ_e = east longitude

h = 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 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)
```

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

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

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.

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

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

$$r_x = p\left[\cos\Omega\cos\left(\omega+\nu\right) - \sin\Omega\cos i\sin\left(\omega+\nu\right)\right]$$
$$r_y = p\left[\sin\Omega\cos\left(\omega+\nu\right) + \cos\Omega\cos i\sin\left(\omega+\nu\right)\right]$$
$$r_z = p\sin i\sin\left(\omega+\nu\right)$$

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

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

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

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

where

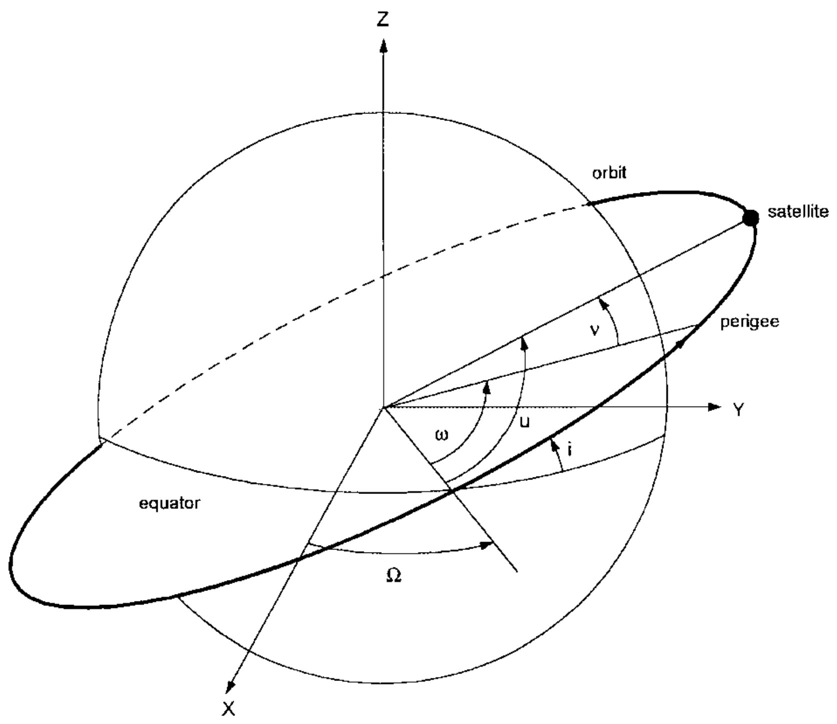
$a =$ semimajor axis
$e =$ orbital eccentricity
$i =$ orbital inclination
$\omega =$ argument of periapsis
$\Omega =$ right ascension of the ascending node
$\nu =$ true anomaly
$p = a\left(1-e^2\right)$

The syntax of this MATLAB function is

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

% convert classical orbital elements to inertial state vector

% 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 periapsis (radians)
%           (0 <= argument of periapsis <= 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)
```

eci2orb.m – convert inertial state vector to six classical orbital elements

This MATLAB function converts an inertial state vector (position and velocity vectors) to six classical orbital elements. The following diagram illustrates the geometry of these orbital elements.



where

$$a = \text{semimajor axis}$$
$$e = \text{orbital eccentricity}$$
$$i = \text{orbital inclination}$$
$$\omega = \text{argument of periapsis}$$
$$\Omega = \text{right ascension of the ascending node}$$
$$\nu = \text{true anomaly}$$

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 ecliptic or the equator. The z-axis of this system is collinear with either the spin axis of the Earth or the north ecliptic pole, and the y-axis completes a right-handed coordinate system.

The orbital inclination is the angle between the fundamental plane and the orbit plane. Orbits with inclinations between 0 and 90 degrees are called *direct* orbits and objects 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 fundamental plane to the ascending node. The argument of periapsis 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 object'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 object'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 the equinoctial orbital elements that 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 perigapsis is determined from

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

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

Finally, the true anomaly is found from the expression

$$\nu = \lambda - \Omega - \omega$$

In these equations, all two argument inverse tangent functions use the four quadrant function `atan3.m` and angular orbital elements which can range from 0 to 360 degrees are also processed with the modulo 2π function, `modulo.m`.

The syntax of this MATLAB function is

```
function oev = eci2orb (r, v)

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

% input

%  r  = inertial position vector (kilometers)
%  v  = inertial 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 periapsis (radians)
%            (0 <= argument of periapsis <= 2 pi)
%  oev(5)  = right ascension of ascending node (radians)
%            (0 <= raan <= 2 pi)
%  oev(6)  = true anomaly (radians)
%            (0 <= true anomaly <= 2 pi)
```

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

This MATLAB function can be used to convert <u>modified</u> 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) = right ascension of ascending node (radians)
%  coev(5) = argument of periapsis (radians)
%  coev(6) = true anomaly (radians)
```

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

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

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

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

right ascension of the ascending node

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

true anomaly

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

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(mee)

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

% 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

%  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

$$\alpha^2 = h^2 - k^2$$

$$s^2 = 1 + h^2 + k^2$$

$$r = \frac{p}{w}$$

$$w = 1 + f\cos L + g\sin L$$

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(reci, veci)

% convert eci state vector to
% modified equinoctial elements

% input

%  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)
```

rotmat.m – fundamental rotation matrix

This MATLAB function computes the elementary rotation matrix for a user-defined rotation angle about a user-defined rotation axis. All calculations follow the right-handed rule and counterclockwise rotation angles are positive.

The three rotation matrices for the x, y and z axes are defined by

$$R_x\left(\theta\right)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & \sin\theta\\ 0 & -\sin\theta & \cos\theta\end{bmatrix}$$

$$R_y\left(\theta\right)=\begin{bmatrix}\cos\theta & 0 & -\sin\theta\\ 0 & 1 & 0\\ \sin\theta & 0 & \cos\theta\end{bmatrix}$$

$$R_z\left(\theta\right)=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

where θ is the rotation angle.

Transformations between fundamental astronomical coordinate systems can also be accomplished using various combinations of these matrices. For example, to transform a position vector between the ecliptic and equatorial coordinate systems, the following matrix-vector relationships can be used:

$$\begin{pmatrix}x\\ y\\ z\end{pmatrix}_{\lambda,\beta}=\mathbf{R}_x\left(\varepsilon\right)\begin{pmatrix}x\\ y\\ z\end{pmatrix}_{\alpha,\delta}$$

$$\begin{pmatrix}x\\ y\\ z\end{pmatrix}_{\alpha,\delta}=\mathbf{R}_x\left(-\varepsilon\right)\begin{pmatrix}x\\ y\\ z\end{pmatrix}_{\lambda,\beta}$$

The matrix-vector form of the transformation between the equatorial and local horizontal coordinate systems are as follows:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{A,a} = \mathbf{R}_z\left(-180^\circ\right)\mathbf{R}_y\left(90^\circ - \phi\right)\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{h,\delta}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{h,\delta} = \mathbf{R}_y\left(\phi - 90^\circ\right)\mathbf{R}_z\left(180^\circ\right)\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{A,a}$$

The syntax of this MATLAB function is

```
function rmatrix = rotmat(iaxis, theta)

% rotation matrix

% input

%  iaxis = 1 ==> x-axis
%        = 2 ==> y-axis
%        = 3 ==> z-axis
%  theta = rotation angle (radians)

% output

%  rmatrix = rotation matrix
```

24. NOVAS Routines

The *Celestial Computing with MATLAB* software suite includes MATLAB versions of several of the Fortran subroutines contained in the NOVAS (Naval Observatory Vector Astrometry Subroutines) software package. This section documents the purpose and calling arguments for these routines. Additional information about NOVAS can be found at the Naval Observatory web site located at

http://aa.usno.navy.mil/AA/software/novas/novas_info.html

An excellent discussion about the algorithms used in the NOVAS software can be found in “Mean and Apparent Place Computations in the New IAU System. III. Apparent, Topocentric, and Astrometric Places of Planets and Stars”, G.H. Kaplan, J.A. Hughes, P.K. Seidelmann, C.A. Smith and B.D. Yallop, *The Astronomical Journal*, Vol. 97, No. 4, pages 1197-1210, 1989.

aberat.m – correction for the aberration of light

```
function pos2 = aberat (pos1, ve, tlight)

% this function corrects position vector for aberration of light.
% algorithm includes relativistic terms.  see murray (1981)
% mon. notices royal ast. society 195, 639-648.

% input

%  pos1   = position vector, referred to origin at center of
%           mass of the earth, components in au
%  ve     = velocity vector of center of mass of the earth,
%           referred to origin at solar system barycenter,
%           components in au/day
%  tlight = light time from body to earth in days
%           if tlight = 0, this function will compute tlight

% output

%   pos2 = position vector, referred to origin at center of
%          mass of the earth, corrected for aberration,
%          components in au
```

etilt1.m – orientation quantities based on DE405

```
function [oblm, oblt, eqeq, dpsi, deps] = etilt1 (tjd)

% this function computes quantities related to the orientation
% of the earth's rotation axis at julian date tjd

% nutation parameters obtained from jpleph

% input

%  tjd = tdb julian date for orientation parameters

% output

%  oblm = mean obliquity of the ecliptic at date tjd (degrees)
%  oblt = true obliquity of the ecliptic at date tjd (degrees)
%  eqeq = equation of the equinoxes at date tjd (arc seconds)
%  dpsi = nutation in longitude at date tjd (arc seconds)
%  deps = nutation in obliquity at date tjd (arc seconds)
```

etilt2.m – orientation quantities based on IAU 1980 nutation

```
function [oblm, oblt, eqeq, dpsi, deps] = etilt2 (tjd)

% this function computes quantities related to the orientation
% of the earth's rotation axis at julian date tjd

% nutation parameters obtained from function nutation

% input

%  tjd = tdb julian date for orientation parameters

% output

%  oblm = mean obliquity of the ecliptic at date tjd (degrees)
%  oblt = true obliquity of the ecliptic at date tjd (degrees)
%  eqeq = equation of the equinoxes at date tjd (arc seconds)
%  dpsi = nutation in longitude at date tjd (arc seconds)
%  deps = nutation in obliquity at date tjd (arc seconds)
```

geocen.m – move coordinates origin to the Earth center-of-mass

```
function [pos2, tlight] = geocen (pos1, pe)

% this function moves the origin of coordinates from the
% barycenter of the solar system to the center of mass of the
% earth, i.e., this function corrects for parallax.

% input

%  pos1 = position vector, referred to origin at solar system
%         barycenter (au)
%  pe   = position vector of center of mass of the earth,
%         referred to origin at solar system barycenter (au)

% output

%   pos2   = position vector, referred to origin at center of
%            mass of the earth (au)
%   tlight = light time from body to earth in days
```

nutate1.m – nutation transformation

```
function pos2 = nutate (tjd, pos1)

% this function nutates equatorial rectangular coordinates from
% mean equator and equinox of epoch to true equator and equinox of
% epoch. see pages 41-45 of the explanatory supplement to the ae.

% source ephemeris is jpleph via etilt1

% input

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

% output

%   pos2 = position vector, geocentric equatorial rectangular
%          coordinates, referred to true equator and equinox
%          of epoch (out)

% note:
% if tjd is negative, inverse nutation (true to mean) is applied
```

nutate2.m – nutation transformation

```
function pos2 = nutate2 (tjd, pos1)

% this function nutates equatorial rectangular coordinates from
% mean equator and equinox of epoch to true equator and equinox of
% epoch. see pages 41-45 of the explanatory supplement to the ae.

% uses nutation function via etilt2

% input

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

% output

%   pos2 = position vector, geocentric equatorial rectangular
%          coordinates, referred to true equator and equinox
%          of epoch (out)

% note:
% if tjd is negative, inverse nutation (true to mean) is applied
```

pnsw.m – Earth-fixed to space-fixed transformation

```
function vecs = pnsw (tjd, gast, x, y, vece)

% transforms a vector from earth-fixed system to space-fixed system
% by applying rotations for wobble, spin, nutation, and precession.
% (combined rotation is symbolized p n s w.) specifically,
% it transforms a vector from earth-fixed geographic system to
% space-fixed system based on mean equator and equinox of j2000.0.

% input

%  tjd  = tdt julian date
%  gast = greenwich apparent sidereal time (hours)
%  x    = conventionally-defined x coordinate of rotational
%         pole with respect to cio (arc seconds)
%  y    = conventionally-defined y coordinate of rotational
%         pole with respect to cio (arc seconds)
%  vece = vector in geocentric rectangular
%         earth-fixed system, referred to geographic
%         equator and greenwich meridian
% output

%  vecs = vector in geocentric rectangular space-fixed system,
%         referred to mean equator and equinox of j2000.0

%  note
%         tjd = 0 means no precession/nutation transformation
%         gast = 0 means no spin transformation
%         x = y = 0 means no wobble transformation
```

propmo.m – apply proper motion correction

```
function pos2 = propmo (tjd1, pos1, vel1, tjd2)

% this function applies proper motion, including
% foreshortening effects, to a star's position.

% input

%  tjd1 = tdb julian date of first epoch
%  pos1 = position vector at first epoch
%  vel1 = velocity vector at first epoch
%  tjd2 = tdb julian date of second epoch

% output

%  pos2 = position vector at second epoch
```

solsys.m – interface to JPL ephemerides

```
function [pos, vel, ierr] = solsys (tjd, body, origin)

% purpose

% this is solsys version '2-da'. it is intended to provide
% an interface between the jpl direct-access solar system
% ephemerides and the 'novas' astrometric function library

% references

% standish, e.m. and newhall, x x (1988). "the jpl
% export planetary ephemeris"; jpl document dated 17 june 1988.

% kaplan, g.h. (1988). "novas: naval observatory vector astrometry
% subroutines"; usno document dated 20 october 1988

% input

%  tjd    = julian date of the desired time, on the tdb time scale
%  body   = body identification number for the
%           solar system object of interest;
%           mercury = 1,...,pluto = 9, sun = 10, moon = 11
%  origin = origin code; solar system barycenter = 0,
%           center of mass of the sun = 1

% output

%  pos  = position vector of 'body' at tjd;
%         equatorial rectangular coordinates in
%         au referred to the mean equator and
%         equinox of j2000.0
% vel  = velocity vector of 'body' at tjd;
%        equatorial rectangular system referred
%        to the mean equator and equinox of
%        j2000.0, in au/day
% ierr = 0 ... everything ok .
%      = 1 ... invalid value of 'body'
%      = 2 ... invalid value of 'origin'
%      = 3 ... error detected by jpl software;
%              tjd out of range
```

spin.m – rotating to non-rotating transformation

```
function pos2 = spin (st, pos1)

% this subroutine transforms geocentric rectangular coordinates
% from rotating system based on rotational equator and orthogonal
% reference meridian to non-rotating system based on true equator
% and equinox of date.

% input

% st   = local apparent sidereal time at reference meridian (hours)
% pos1 = vector in geocentric rectangular rotating system, referred
%        to rotational equator and orthogonal reference meridian

% output

% pos2 = vector in geocentric rectangular non-rotating system,
%        referred to true equator and equinox of date
```

sunfld.m – correct for the deflection of light

```
function pos2 = sunfld (pos1, pe)

% this function corrects position vector for the deflection
% of light in the gravitational field of the sun. see murray (1981)
% mon. notices royal ast. society 195, 639-648. this function valid
% for bodies within the solar system as well as for stars.

% input

% pos1 = position vector, referred to origin at center of mass
%        of the earth, components in au
% pe   = position vector of center of mass of the earth,
%        referred to origin at center of mass of the sun,
%        components in au

% output

% pos2 = position vector, referred to origin at center of mass
%        of the earth, corrected for gravitational deflection,
%        components in au
```

tdtimes.m – compute TDT julian date

```
function [tdtjd, secdif] = tdtimes (tdbjd)

% this function computes the terrestrial dynamical
% time (tdt) julian date corresponding to a barycentric
% dynamical time (tdb) julian date. expressions used in
% this version are approximations resulting in accuracies
% of about 20 microseconds.

% input

%  tdbjd = tdb julian date

% output

%  tdtjd = tdt julian date

%  secdif = difference tdbjd-tdtjd, in seconds (out)
```

terra.m – position and velocity vectors of a terrestrial observer

```
function [pos, vel] = terra (glon, glat, ht, st)

% this functions computes the position and velocity vectors of
% a terrestrial observer with respect to the center of the earth.

% input

%  glon = longitude of observer with respect to reference
%         meridian (east +) in degrees
%  glat = geodetic latitude (north +) of observer in degrees
%  ht   = height of observer in meters
%  st   = local apparent sidereal time at reference meridian
%         in hours

% output

%  pos = position vector of observer with respect to center
%        of earth, equatorial rectangular coordinates,
%        referred to true equator and equinox of date,
%        components in au
%  vel = velocity vector of observer with respect to center
%        of earth, equatorial rectangular coordinates,
%        referred to true equator and equinox of date,
%        components in au/day

%  note: if reference meridian is greenwich and st=0, pos
%        is effectively referred to equator and greenwich
```

vectrs.m – convert angular quantities to vectors

```
function [pos, vel] = vectrs (ra, dec, pmra, pmdec, parllx, rv)

% this function converts angular quantities to vectors

% input

%  ra     = right ascension (hours)
%  dec    = declination (degrees)
%  pmra   = proper motion in ra (seconds of time per julian century)
%  pmdec  = proper motion in dec per julian century (seconds of arc)
%  parllx = parallax (seconds of arc)
%  rv     = radial velocity (kilometers per second)

% output

%  pos = position vector, equatorial rectangular coordinates (au)
%  vel = velocity vector, equatorial rectangular coordinates (au/day)
```

wobble.m – polar motion correction

```
function pos2 = wobble (x, y, pos1)

% this subroutine corrects earth-fixed geocentric rectangular
% coordinates for polar motion. it transforms a vector from
% earth-fixed geographic system to rotating system based on
% rotational equator and orthogonal greenwich meridian through
% axis of rotation.

% input

%  x    = conventionally-defined x coordinate of rotational
%         pole with respect to cio (arc seconds)
%  y    = conventionally-defined y coordinate of rotational
%         pole with respect to cio (arc seconds)
%  pos1 = vector in geocentric rectangular earth-fixed system,
%         referred to geographic equator and greenwich meridian

% output

%  pos2 = vector in geocentric rectangular rotating system,
%         referred to rotational equator and orthogonal greenwich meridian
```

25. Bibliography and References

Bonavito, N. L. and Der, G. J., *Orbital and Celestial Mechanics*, AIAA Progress in Astronautics and Aeronautics, Volume 177, 1998.

Bretagnon, P., and Simon, J., *Planetary Programs and Tables from –4000 to +2800,* Willmann-Bell, 1986.

Brouwer, D., and Clemence, G., *Methods of Celestial Mechanics*, Academic Press, 1961.

Brumberg, V. A., *Analytical Techniques of Celestial Mechanics*, Springer-Verlag, 1995.

Chapront, J., and Chapront-Touze, M., *Lunar Tables and Programs from 4000 B.C. to A.D. 8000*, Willmann-Bell, 1991.

Chebotarev, G. A., *Analytical and Numerical Methods of Celestial Mechanics,* American Elsevier Publishing Company, 1967.

Danby, J. M. A., *Fundamentals of Celestial Mechanics*, Willmann-Bell, 1988.

Dormand, J. R., *Numerical Methods for Differential Equations*, CRC Press, 1996.

Dubyago, A. D., *The Determination of Orbits*, The Macmillian Company, 1961.

Escobal, P. R., *Methods of Orbit Determination*, Robert E. Krieger Publishing, 1976.

Escobal, P. R., *Methods of Astrodynamics*, Robert E. Krieger Publishing, 1979.

Forsythe, G. E., Malcolm, M. A., and Moler, C. B., *Computer Methods for Mathematical Computations*, Prentice-Hall, 1977.

Heafner, P. J., *Fundamental Ephemeris Computations*, Willmann-Bell, 1999.

Meeus, J., *Astronomical Algorithms*, Willmann-Bell, 1991.

Meeus, J., *Tables of the Sun, Moon and Planets*, Willmann-Bell, 1995.

Meeus, J., *Transits*, Willmann-Bell, 1989.

Meeus, J., *Mathematical Astronomy Morsels*, Willmann-Bell, 1997.

Montenbruck, O., and Pfleger, T., *Astronomy on the Personal Computer*, Springer-Verlag, 1994.

Moulton, F. R., *An Introduction to Celestial Mechanics*, Dover, 1970.

Seidelmann, P. K. (Editor), *Explanatory Supplement to the Astronomical Alamanac*, University Science Books, 1992.

Szebehely, V. G., *Adventures in Celestial Mechanics*, University of Texas Press, 1989.