

A MATLAB Script for Predicting Orbital Events of the Moon

This document describes a MATLAB script named `levents.m` that can be used to compute important orbital events of the Moon. The motion of the Moon is modeled using the JPL DE421 binary ephemeris and this script uses routines from the MICE software suite to read and evaluate this ephemeris. The MICE routines and ephemeris file are available at naif.jpl.nasa.gov/naif/toolkit_MATLAB.html. MICE is a MATLAB implementation of the SPICE library created by JPL.

Using a combination of one-dimensional minimization and root-finding, the `levents` MATLAB script computes the following geocentric characteristics of the Moon's orbital motion.

- 1) apogee and perigee
- 2) minimum and maximum geocentric declination
- 3) user-defined geocentric right ascension
- 4) user-defined geocentric declination
- 5) user-defined true anomaly
- 6) minimum and maximum orbital eccentricity
- 7) nodal crossings

Script options 1, 2, and 6 are minimum and maximum conditions of the lunar orbit. The nodal crossings correspond to times when the Moon crosses the Earth's equatorial plane.

Interacting with the script

The `levents` MATLAB script will prompt you for the calendar date at which to start the events search along with a search duration in days. The following illustrates these two prompts along with typical user inputs shown in bold font. Please be sure to provide all digits of the calendar year.

```
program levents

< orbital events of the moon >

please input the initial calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,2014

please input the search duration (days)
? 30
```

The script will then display a list of event options with the following menu;

```
please select the lunar event to predict

<1> apogee and perigee
<2> minimum and maximum geocentric declination
<3> user-defined geocentric right ascension
<4> user-defined geocentric declination
```

Orbital Mechanics with MATLAB

<5> user-defined true anomaly

<6> minimum and maximum orbital eccentricity

<7> nodal crossings

?

For menu option 3, the script will also prompt for the user-defined value of geocentric right ascension with the following display;

```
please input the geocentric right ascension (degrees)
(0 <= right ascension <= 360)
?
```

For menu option 4, the script will also prompt for the user-defined value of geocentric declination with the following display;

```
please input the geocentric declination (degrees)
(-90 <= declination <= +90)
```

For menu option 5, the script will also prompt for the user-defined value of true anomaly with the following display;

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

For each of these script options, the input unit is degrees. Please note the range of valid input.

Script example

The following is a typical user interaction with this MATLAB script along with the program output for the minimum and maximum geocentric declination of the moon during January 2014.

```
program levents

< orbital events of the moon >

please input the initial calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,2014

please input the search duration (days)
? 30

please select the lunar event to predict

<1> apogee and perigee
<2> minimum and maximum geocentric declination
<3> user-defined geocentric right ascension
<4> user-defined geocentric declination
<5> user-defined true anomaly
<6> minimum and maximum orbital eccentricity
```

Orbital Mechanics with MATLAB

<7> nodal crossings

? 2

time and conditions at minimum declination

=====

```
calendar date          27-Jan-2014
TDB time              16:55:48.131
TDB Julian date       2456685.20541819
UTC time              16:54:41.947
UTC Julian date       2456685.20465216
geocentric declination -19.38236906 degrees
```

geocentric orbital elements and state vector
(Earth mean equator and equinox J2000)

sma (km)	eccentricity	inclination (deg)	argper (deg)
+3.88526254363298e+05	+8.01745529529702e-02	+1.93823690622682e+01	+3.10295086963237e+02
raan (deg)	true anomaly (deg)	arglat (deg)	period (days)
+3.51862407979640e+02	+3.19704913531811e+02	+2.70000000495048e+02	+2.78950919892399e+01
rx (km)	ry (km)	rz (km)	rmag (km)
-4.85753538318047e+04	-3.39710302374387e+05	-1.20729016559839e+05	+3.63783108429581e+05
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
+1.07446967239062e+00	-1.03431188427361e-01	+1.74857394845693e-02	+1.07957808365677e+00

time and conditions at maximum declination

=====

```
calendar date          09-Feb-2014
TDB time              15:50:13.403
TDB Julian date       2456698.15987735
UTC time              15:49:07.218
UTC Julian date       2456698.15911132
geocentric declination 19.29480971 degrees
```

geocentric orbital elements and state vector
(Earth mean equator and equinox J2000)

sma (km)	eccentricity	inclination (deg)	argper (deg)
+3.86033424966801e+05	+5.33702230739854e-02	+1.92948097076573e+01	+3.01922689183831e+02
raan (deg)	true anomaly (deg)	arglat (deg)	period (days)
+3.52061507031174e+02	+1.48077310699116e+02	+8.99999998829470e+01	+2.76270558739407e+01
rx (km)	ry (km)	rz (km)	rmag (km)
+5.25578732820930e+04	+3.76904041635659e+05	+1.33228329807996e+05	+4.03198182670332e+05
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
-9.58447892208817e-01	+1.61018650533875e-01	+9.48906430849668e-03	+9.71925619705914e-01

Additional script examples can be found in Appendix A.

Orbital Mechanics with MATLAB

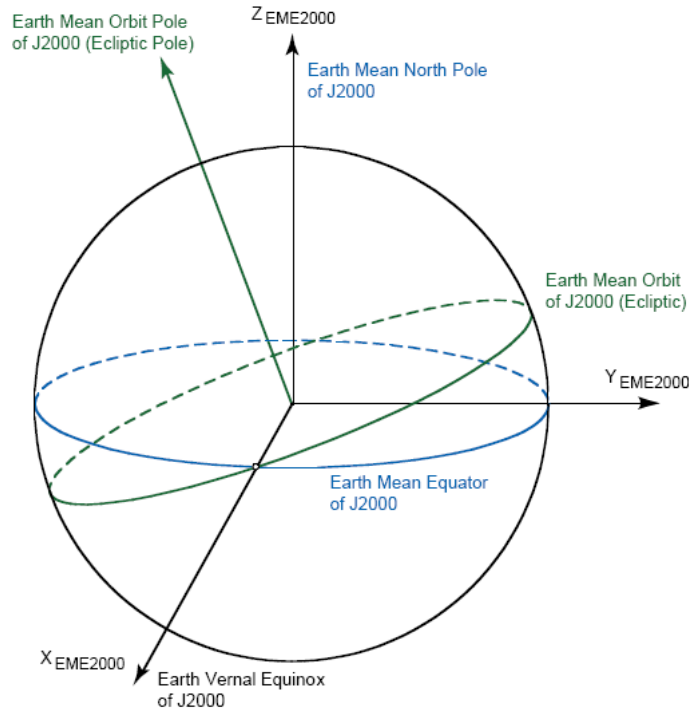
The orbital event summary screen contains the following information:

```
calendar date = calendar date of trajectory event
TDB time = TDB time of trajectory event
TDB Julian date = Julian Date of trajectory event on TDB time scale
UTC time = UTC time of trajectory event
UTC Julian date = Julian Date of trajectory event on UTC time scale
sma (km) = semimajor axis in kilometers
eccentricity = orbital eccentricity (non-dimensional)
inclination (deg) = orbital inclination in degrees
argper (deg) = argument of perigee in degrees
raan (deg) = right ascension of the ascending node in degrees
true anomaly (deg) = true anomaly in degrees
arglat (deg) = argument of latitude in degrees. The argument of latitude is the sum of
    true anomaly and argument of perigee.
period (days) = orbital period in days
rx (km) = x-component of the Moon's position vector in kilometers
ry (km) = y-component of the Moon's position vector in kilometers
rz (km) = z-component of the Moon's position vector in kilometers
rmag (km) = scalar magnitude of the Moon's position vector in kilometers
vx (kps) = x-component of the Moon's velocity vector in kilometers per second
vy (kps) = y-component of the Moon's velocity vector in kilometers per second
vz (kps) = z-component of the Moon's velocity vector in kilometers per second
vmag (kps) = scalar magnitude of the Moon's velocity vector in kilometers per second
```

Technical Discussion

In this MATLAB script, the orbital motion of the Moon is evaluated in the Earth mean equator and equinox of J2000 (EME2000) coordinate system. The following figure illustrates the geometry of the EME2000 coordinate system. The origin of this ECI inertial coordinate system is the geocenter and the fundamental plane is the Earth's mean equator. The z-axis of this system is normal to the Earth's mean equator at epoch J2000, the x-axis is parallel to the vernal equinox of the Earth's mean orbit at epoch J2000, and the y-axis completes the right-handed coordinate system. The epoch J2000 is the Julian Date 2451545.0 which corresponds to January 1, 2000, 12 hours Terrestrial Time (TT).

Orbital Mechanics with MATLAB



MATLAB functions

In this MATLAB script, the geocentric position and velocity vectors of the Moon are computed using a function named `jpleph_mice` which requires initialization the first time it is called. The proper initialization for this function is as follows:

```
iephem = 1;  
ephname = 'de421.bsp';  
km = 1;
```

The second item is the name of the binary ephemeris file to use for all calculations. The third item determines the units of the output. If `km = 1`, the output will be in the units of kilometers and kilometers per second. If `km = 0`, the output will be in the units of Astronomical Units and Astronomical Units per day. These three items should be placed in a `global` statement at the beginning of the main script which calls either of these functions.

Note that the value of the Astronomical Unit, in kilometers, used in a particular JPL ephemeris is available as the constant `au` which is placed in `global` by the main script. The actual value used in a particular JPL ephemeris can be found in the header file posted on the JPL website. For example, the following are the first three lines in the `header.421` data file. The value of Astronomical Unit used in this ephemeris is the first number in row three of this data file.

```
0.421000000000000000D+03 0.421000000000000000D+03 0.0000000000000000D+00  
0.120080211181117000D+17 0.0000000000000000D+00 0.2997924580000000D+06  
0.149597870699626200D+09 0.813005690699153000D+02 0.491254957186794000D-10
```

The following is the syntax for the `jpleph_mice` MATLAB function:

Orbital Mechanics with MATLAB

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

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

% input

%   et       = TDB julian 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
%           6 = saturn
%           7 = uranus

% output

%   rrd = output 6-word array containing position and velocity
%         of point 'ntarg' relative to 'ncent'. the units are
%         determined by the value of km passed via global.

% global

%   iephem = initialization flag (1 = initialize)
%   ephname = name of ephemeris binary data file (de421.bsp, etc.)
%   km      = state vector units flag (1 = km & km/sec, 0 = au & au/day)
%   au      = numerical value of astronomical unit (kilometers)
```

It is good programming practice to close the binary ephemeris file at the end of the main script with the following statement:

```
% unload ephemeris

cspice_unload('de421.bsp');
```

Time systems

Coordinated Universal Time, UTC

Coordinated Universal Time (UTC) is the time scale available from broadcast time signals. It is a compromise between the highly stable atomic time and the irregular earth rotation. UTC is the international basis of civil and scientific time.

Terrestrial Time, TT

Terrestrial Time is the time scale that would be kept by an ideal clock on the geoid - approximately, sea level on the surface of the Earth. Since its unit of time is the SI (atomic) second, TT is independent of the variable rotation of the Earth. TT is meant to be a smooth and continuous “coordinate” time scale independent of Earth rotation. In practice TT is derived from International Atomic Time (TAI), a time scale kept by real clocks on the Earth's surface, by the relation $\mathbf{TT} = \mathbf{TAI} + 32^{\text{s}}.184$. It is the time scale now used for the precise calculation of future astronomical events observable from Earth.

$$\mathbf{TT} = \mathbf{TAI} + 32.184 \text{ seconds}$$

$$\mathbf{TT} = \mathbf{UTC} + (\text{number of leap seconds}) + 32.184 \text{ seconds}$$

Barycentric Dynamical Time, TDB

Barycentric Dynamical Time is the time scale that would be kept by an ideal clock, free of gravitational fields, co-moving with the solar system barycenter. It is always within 2 milliseconds of TT, the difference caused by relativistic effects. TDB is the time scale now used for investigations of the dynamics of solar system bodies.

$$\mathbf{TDB} = \mathbf{TT} + \text{periodic corrections}$$

where typical periodic corrections (USNO Circular 179) are

$$\begin{aligned} \mathbf{TDB} = \mathbf{TT} &+ 0.001657 \sin(628.3076T + 6.2401) \\ &+ 0.000022 \sin(575.3385T + 4.2970) \\ &+ 0.000014 \sin(1256.6152T + 6.1969) \\ &+ 0.000005 \sin(606.9777T + 4.0212) \\ &+ 0.000005 \sin(52.9691T + 0.4444) \\ &+ 0.000002 \sin(21.3299T + 5.5431) \\ &+ 0.000010T \sin(628.3076T + 4.2490) + \dots \end{aligned}$$

In this equation, the coefficients are in seconds, the angular arguments are in radians, and T is the number of Julian centuries of TT from J2000; $T = (\text{Julian Date}(TT) - 2451545.0) / 36525$.

The following is the MATLAB source code for the routine ported from the NOVAS Fortran subroutine. Notice that the NOVAS name was simply `times` and the ported version is named `novas_times` to avoid confusion with the built-in MATLAB function.

```
function [ttjd, secdif] = novas_times (tdbjd)

% this function computes the terrestrial time (tt) julian date
% corresponding to a barycentric dynamical time (tdb) julian date.
% the expression used in this version is a truncated form of a
% longer and more precise series given by fairhead & bretagnon
% (1990) a&a 229, 240. the result is good to about 10 microseconds.

% input
```

Orbital Mechanics with MATLAB

```
% tdbjd = tdb julian date

% output

% ttjd = tt julian date

% secdif = difference tdbjd - ttjd, in seconds

% ported from NOVAS 3.0

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

% t0 = tdb julian date of epoch j2000.0 (tt)

t0 = 2451545.0d0;

t = (tdbjd - t0) / 36525.0d0;

% expression given in usno circular 179, eq. 2.6

secdif = 0.001657d0 * sin(628.3076d0 * t + 6.2401d0) ...
+ 0.000022d0 * sin(575.3385d0 * t + 4.2970d0) ...
+ 0.000014d0 * sin(1256.6152d0 * t + 6.1969d0) ...
+ 0.000005d0 * sin(606.9777d0 * t + 4.0212d0) ...
+ 0.000005d0 * sin(52.9691d0 * t + 0.4444d0) ...
+ 0.000002d0 * sin(21.3299d0 * t + 5.5431d0) ...

+ 0.000010d0 * t * sin(628.3076d0 * t + 4.2490d0);

ttjd = tdbjd - secdif / 86400.0d0;
```

The fundamental time argument for the lunar ephemeris function used in this MATLAB script is “ephemeris” time. As implemented here, we assume this time argument to be Barycentric Dynamical Time (TDB). To report the time of these celestial events in Universal Coordinated Time (UTC) or civil time, we need an algorithm to make this time conversion.

The following is the MATLAB source code for a function which iteratively performs this calculation using Brent’s root-finding method.

```
function jdutc = tdb2utc (jdtddb)

% convert TDB julian date to UTC julian date

% input

% jdtddb = TDB julian date

% output

% jdutc = UTC julian date

% Orbital Mechanics with MATLAB

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

global jdsaved
```


Orbital Mechanics with MATLAB

```
jdsaved = jdtdb;

% convergence tolerance
rtol = 1.0d-8;

% set lower and upper bounds
x1 = jdsaved - 0.1;
x2 = jdsaved + 0.1;

% solve for UTC julian date using Brent's method
[xroot, froot] = brent ('jdfunc', x1, x2, rtol);

jdutc = xroot;

end
```

This function calls the following MATLAB *objective function* during the calculations.

```
function fx = jdfunc (jdin)

% objective function for tdb2utc

% input

%   jdin = current value for UTC julian date

% output

%   fx = delta julian date

% Orbital Mechanics with MATLAB

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

global jdsaved

tai_utc = findleap(jdin);

fx = utc2tdb (jdin, tai_utc) - jdsaved;

end
```

Notice that this function requires the `findleap` function which calculates the number of leap seconds for the current UTC Julian date value. The `jdfunc` function is computing the difference between the TDB Julian date input by the user and the value computed by the `utc2tdb` MATLAB function. The algorithm has converged whenever this value is less than or equal to the user-defined tolerance `rtol`.

Leap seconds calculation

The difference between International Atomic Time (TAI) and Universal Coordinated Time (UTC) is the number of current leap seconds. International Atomic Time (TAI, Temps Atomique International) is a physical time scale with the unit of the SI (System International) second and derived from a statistical

Orbital Mechanics with MATLAB

timescale based on a large number of atomic clocks. Coordinated Universal Time (UTC) is the time scale available from broadcast time signals. It is a compromise between the highly stable atomic time and the irregular earth rotation. UTC is the international basis of civil and scientific time.

The calculation of leap seconds in this MATLAB script is performed by a function that reads a simple ASCII data file and evaluates the current value of leap seconds. The leap second function must be initialized by including the following statements in the main script.

```
% read leap seconds data file

readleap;
```

The `readleap` MATLAB function reads the contents of the following simple comma-separated-variable (csv) two column data file. The name of this file is `tai-utc.dat`.

```
2441317.5, 10.0
2441499.5, 11.0
2441683.5, 12.0
2442048.5, 13.0
2442413.5, 14.0
2442778.5, 15.0
2443144.5, 16.0
2443509.5, 17.0
2443874.5, 18.0
2444239.5, 19.0
2444786.5, 20.0
2445151.5, 21.0
2445516.5, 22.0
2446247.5, 23.0
2447161.5, 24.0
2447892.5, 25.0
2448257.5, 26.0
2448804.5, 27.0
2449169.5, 28.0
2449534.5, 29.0
2450083.5, 30.0
2450630.5, 31.0
2451179.5, 32.0
2453736.5, 33.0
2454832.5, 34.0
```

The first column of this data file is the Julian date, on the UTC time scale, at which the leap second became valid. The second column is the leap second value, in seconds.

Note that this data is passed between the leap second MATLAB functions by way of a global statement.

```
global jdateleap leapsec
```

The MATLAB function that actually reads and evaluates the current value of leap seconds has the following syntax and single argument.

```
function leapsecond = findleap(jdate)

% find number of leap seconds for utc julian date
```

Orbital Mechanics with MATLAB

```
% input

% jdate = utc julian date

% input via global

% jdateleap = array of utc julian dates
% leapsec    = array of leap seconds

% output

% leapsecond = number of leap seconds
```

The leap seconds data file should be updated whenever the International Earth Rotation and Reference Systems Service (IERS) announces a new leap second.

Appendix A

Additional Script Examples

This appendix includes output created by several of the other `levents` script options.

```
program levents

< orbital events of the moon >

please input the initial calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 1,1,2014

please input the search duration (days)
? 30

please select the lunar event to predict

<1> apogee and perigee

<2> minimum and maximum geocentric declination

<3> user-defined geocentric right ascension

<4> user-defined geocentric declination

<5> user-defined true anomaly

<6> minimum and maximum orbital eccentricity

<7> nodal crossings

? 3

please input the geocentric right ascension (degrees)
(0 <= right ascension <= 360)
? 120

time and conditions at user-defined right ascension
=====

calendar date          16-Jan-2014
TDB time               11:15:07.295
TDB Julian date        2456673.96883443
UTC time               11:14:01.110
UTC Julian date        2456673.96806841

geocentric orbital elements and state vector
(Earth mean equator and equinox J2000)
-----

      sma (km)      eccentricity      inclination (deg)      argper (deg)
+3.92030629436879e+05 +3.69860119614338e-02 +1.94692410906878e+01 +3.02260247507493e+02

      raan (deg)      true anomaly (deg)      arglat (deg)      period (days)
+3.51692297464799e+02 +1.84418516689778e+02 +1.26678764197271e+02 +2.82733481403732e+01

      rx (km)      ry (km)      rz (km)      rmag (km)
-1.95846297076435e+05 +3.39215737025722e+05 +1.08655677168906e+05 +4.06483879760918e+05

      vx (kps)      vy (kps)      vz (kps)      vmag (kps)
-8.48917032439884e-01 -4.31347035509562e-01 -1.94249162125556e-01 +9.71829784476637e-01
```

Orbital Mechanics with MATLAB

geocentric right ascension 120.00000000 degrees

program levents

< orbital events of the moon >

please input the initial calendar date

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

? **1,1,2014**

please input the search duration (days)

? **30**

please select the lunar event to predict

<1> apogee and perigee

<2> minimum and maximum geocentric declination

<3> user-defined geocentric right ascension

<4> user-defined geocentric declination

<5> user-defined true anomaly

<6> minimum and maximum orbital eccentricity

<7> nodal crossings

? **6**

time and conditions at minimum eccentricity

=====

calendar date	16-Jan-2014
TDB time	04:52:54.743
TDB Julian date	2456673.70341138
UTC time	04:51:48.559
UTC Julian date	2456673.70264536
orbital eccentricity	0.03693123 degrees

geocentric orbital elements and state vector

(Earth mean equator and equinox J2000)

sma (km)	eccentricity	inclination (deg)	argper (deg)
+3.92053087994261e+05	+3.69312337246207e-02	+1.94726115663969e+01	+3.02140356572455e+02
raan (deg)	true anomaly (deg)	arglat (deg)	period (days)
+3.51677918019766e+02	+1.81410918479402e+02	+1.23551275051857e+02	+2.82757777504264e+01
rx (km)	ry (km)	rz (km)	rmag (km)
-1.76083175319990e+05	+3.48573938945866e+05	+1.12939911436396e+05	+4.06527365792596e+05
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
-8.74212262361963e-01	-3.84589848574662e-01	-1.79290695107703e-01	+9.71751812266335e-01

time and conditions at maximum eccentricity

=====

Orbital Mechanics with MATLAB

```
calendar date      30-Jan-2014
TDB time           20:39:04.380
TDB Julian date    2456688.36046737
UTC time           20:37:58.196
UTC Julian date    2456688.35970134
orbital eccentricity 0.08890970 degrees
```

geocentric orbital elements and state vector
(Earth mean equator and equinox J2000)

sma (km)	eccentricity	inclination (deg)	argper (deg)
+3.91932987684056e+05	+8.89096980310241e-02	+1.93622366218224e+01	+3.10956166896728e+02
raan (deg)	true anomaly (deg)	arglat (deg)	period (days)
+3.51994152637177e+02	+6.55117989479818e+00	+3.17507346791526e+02	+2.82627858762572e+01
rx (km)	ry (km)	rz (km)	rmag (km)
+2.29164286094128e+05	-2.62160869103048e+05	-8.00133662056496e+04	+3.57276825558916e+05
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
+8.50430760916976e-01	+6.47881277899118e-01	+2.67078849953152e-01	+1.10195904708765e+00

program levents

< orbital events of the moon >

please input the initial calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? **1,1,2014**

please input the search duration (days)
? **30**

please select the lunar event to predict

- <1> apogee and perigee
- <2> minimum and maximum geocentric declination
- <3> user-defined geocentric right ascension
- <4> user-defined geocentric declination
- <5> user-defined true anomaly
- <6> minimum and maximum orbital eccentricity
- <7> nodal crossings

? **7**

time and conditions at ascending node
=====

```
calendar date      06-Jan-2014
TDB time           06:20:10.745
TDB Julian date    2456663.76401325
UTC time           06:19:04.561
```

Orbital Mechanics with MATLAB

UTC Julian date 2456663.76324723

geocentric orbital elements and state vector
(Earth mean equator and equinox J2000)

```
-----
      sma (km)      eccentricity      inclination (deg)      argper (deg)
+3.85971581783473e+05 +7.29960147093920e-02 +1.94780509815064e+01 +2.94421396272985e+02

      raan (deg)      true anomaly (deg)      arglat (deg)      period (days)
+3.51369706512152e+02 +6.55786037242167e+01 +3.59999999997201e+02 +2.76204172908247e+01

      rx (km)      ry (km)      rz (km)      rmag (km)
+3.68448269004108e+05 -5.59218397063176e+04 -6.06983962825325e-06 +3.72667920658947e+05

      vx (kps)      vy (kps)      vz (kps)      vmag (kps)
+2.15458420574909e-01 +9.68255200707477e-01 +3.50017330824906e-01 +1.05188050489184e+00
```

time and conditions at descending node
=====

calendar date 20-Jan-2014

TDB time 22:25:09.075

TDB Julian date 2456678.43413281

UTC time 22:24:02.890

UTC Julian date 2456678.43336679

geocentric orbital elements and state vector
(Earth mean equator and equinox J2000)

```
-----
      sma (km)      eccentricity      inclination (deg)      argper (deg)
+3.86990262729164e+05 +5.03492563329996e-02 +1.93974648943419e+01 +3.08657935121614e+02

      raan (deg)      true anomaly (deg)      arglat (deg)      period (days)
+3.51811237252316e+02 +2.31342064875707e+02 +1.79999999997321e+02 +2.77298357719506e+01

      rx (km)      ry (km)      rz (km)      rmag (km)
-3.94480609837439e+05 +5.67665516047724e+04 +6.18902717874903e-06 +3.98544091561793e+05

      vx (kps)      vy (kps)      vz (kps)      vmag (kps)
-9.26836051146707e-02 -9.24576725110698e-01 -3.26877925301143e-01 +9.85028704821669e-01
```