Rise and Set of the Sun, Moon and Planets

This document describes a MATLAB script named `riseset.m` that can be used to determine 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 of celestial bodies. The source ephemeris for this routine is a JPL binary ephemeris file.

This application uses several functions ported to MATLAB from the Fortran version of the NOVAS (Naval Observatory Vector Astrometry Subroutines) source code developed at the United States Naval Observatory. Information about NOVAS can be found at www.usno.navy.mil/USNO/astronomical-applications/software-products/novas. The `riseset` MATLAB script also uses routines from the MICE software suite to read and evaluate the `de421.bsp` binary ephemeris file. This ephemeris file, platform dependent versions of the MICE mex file, and other MATLAB functions are available at naif.jpl.nasa.gov/naif/toolkit_MATLAB.html. MICE is a MATLAB implementation of the SPICE library created by JPL.

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 orbital events

% time argument in days

% 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 `1.0e-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 = elapsed simulation time (days)

% output

%  fx = objective function at x

% Celestial Computing with MATLAB

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

global tai_utc jdatei itarg icent rootflg

global obslat glon glat ht stlocl

global aunit elevation azimuth

rtd = 180 / pi;

dtr = pi / 180;

jdutc = jdatei + x;

% compute tdt julian date

jdtdt = utc2tdt(jdutc, tai_utc);

[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 minimization/root-finding orbital 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 real 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 real root in the interval because the numerical value of 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 during the search.

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.

User interaction with the script

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. The screen output created by the script illustrates two rise, maximum elevation and set events of the Moon. The initial calendar date was December 2, 2012, and the observer was located at the Chamberlin Observatory in Denver, Colorado. The calendar date and time displayed is on the UTC time scale.

The `riseset` MATLAB script will also display the calendar date, time and topocentric conditions at maximum elevation along with the total duration of the rise-set event.

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

please input the initial UTC calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? 12,2,2012

please input the search duration (days)
? 2

    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 observer
(-90 <= degrees <= +90, 0 <= minutes <= 60, 0 <= seconds <= 60)
(north latitude is positive, south latitude is negative)
? 39,40,36

please input the geographic longitude of the observer
(0 <= degrees <= 360, 0 <= minutes <= 60, 0 <= seconds <= 60)
(east longitude is positive, west longitude is negative)
? -104,57,12

please input the altitude of the observer (meters)
(positive above sea level, negative below sea level)
? 1644


rise and set of the sun, moon and planets
=========================================

    'Moon'


observer latitude        +39d 40m 36.00s

observer east longitude  -104d 57m 12.00s

observer altitude           1644.0000 meters


rise conditions
---------------

calendar date        02-Dec-2012

universal time       02:28:34.529

UTC Julian date      2456263.6032

topocentric azimuth angle    +66d 19m 01.74s
```

```
topocentric elevation angle  +00d -48m 49.99s

maximum elevation conditions
----------------------------

calendar date        02-Dec-2012

universal time       09:43:46.903

UTC Julian date      2456263.9054

topocentric azimuth angle    +179d 28m 15.15s

topocentric elevation angle  +67d 19m 52.94s

set conditions
--------------

calendar date        02-Dec-2012

universal time       16:54:51.079

UTC Julian date      2456264.2048

topocentric azimuth angle    +291d 40m 49.45s

topocentric elevation angle  +00d -48m 53.37s


celestial event duration     +14h 26m 16.5501s


rise conditions
---------------

calendar date        03-Dec-2012

universal time       03:25:53.673

UTC Julian date      2456264.6430

topocentric azimuth angle    +70d 00m 02.23s

topocentric elevation angle  +00d -48m 56.26s

maximum elevation conditions
----------------------------

calendar date        03-Dec-2012

universal time       10:29:52.114

UTC Julian date      2456264.9374

topocentric azimuth angle    +179d 18m 39.63s

topocentric elevation angle  +64d 19m 55.50s

set conditions
--------------

calendar date        03-Dec-2012


universal time       17:29:13.699
```

```
UTC Julian date      2456265.2286

topocentric azimuth angle    +287d 29m 17.32s

topocentric elevation angle  +00d -49m 00.71s


celestial event duration     +14h 03m 20.0267s
```

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 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
 2456109.5,  35.0
 2457204.5,  36.0
 2457754.5,  37.0
```

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

% 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.