SUNRISET.C

```
/* if you have the SOFA C lib installed
/* gcc -o sunriset sunriset.c -lm -lsofa_c

SUNRISET.C - computes Sun rise/set times, start/end of twilight, and
             the length of the day at any date and latitude

Written as DAYLEN.C, 1989-08-16

Modified to SUNRISET.C, 1992-12-01

(c) Paul Schlyter, 1989, 1992

Released to the public domain by Paul Schlyter, December 1992

*/

// #include <sofa.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

/* 1.1.1970 = JD 2440587.5 */
#define EJD (double) 2440587.5

#define J2000 (double) 2451545.0

/* A macro to compute the number of days elapsed since 2000 Jan 0.0 */
/* (which is equal to 1999 Dec 31, 0h UT)                           */

#define days_since_2000_Jan_0(y,m,d) \
    (367L * (y) - ((7 * ((y) + (((m) + 9) / 12))) / 4) + ((275 * (m))  /9) + (d) -
730531.5L)

/* Some conversion factors between radians and degrees */

#ifndef PI
  #define PI        3.1415926535897932384
#endif

#define RADEG     ( 180.0 / PI )
#define DEGRAD    ( PI / 180.0 )
```

```
/* The trigonometric functions in degrees */

#define sind(x)  sin((x) * DEGRAD)
#define cosd(x)  cos((x) * DEGRAD)
#define tand(x)  tan((x) * DEGRAD)

#define atand(x)    (RADEG * atan(x))
#define asind(x)    (RADEG * asin(x))
#define acosd(x)    (RADEG * acos(x))
#define atan2d(y,x) (RADEG * atan2(y,x));


/* Following are some macros around the "workhorse" function __daylen__    */
/* They mainly fill in the desired values for the reference altitude       */
/* below the horizon, and also selects whether this altitude should        */
/* refer to the Sun's center or its upper limb.                            */
/* This macro computes the length of the day, from sunrise to sunset.      */
/* Sunrise/set is considered to occur when the Sun's upper limb is         */
/* 35 arc minutes below the horizon (this accounts for the refraction      */
/* of the Earth's atmosphere).                                             */
#define day_length(jd, lon, lat)  \
        __daylen__( jd, lon, lat, -50.0/60.0, 1 );
// #define day_length(year,month,day,lon,lat)  \
//         __daylen__( year, month, day, lon, lat, -50.0/60.0, 1 )

/* This macro computes the length of the day, including civil twilight.    */
/* Civil twilight starts/ends when the Sun's center is 6 degrees below     */
/* the horizon.                                                            */
#define day_civil_twilight_length(jd,lon,lat)  \
        __daylen__( jd, lon, lat, -6.0, 0 );
// #define day_civil_twilight_length(year,month,day,lon,lat)  \
//         __daylen__( year, month, day, lon, lat, -6.0, 0 )

/* This macro computes the length of the day, incl. nautical twilight.     */
/* Nautical twilight starts/ends when the Sun's center is 12 degrees       */
/* below the horizon.                                                      */
#define day_nautical_twilight_length(jd, lon, lat)  \
        __daylen__( jd, lon, lat, -12.0, 0 );
// #define day_nautical_twilight_length(year,month,day,lon,lat)  \
//         __daylen__( year, month, day, lon, lat, -12.0, 0 )

/* This macro computes the length of the day, incl. astronomical twilight. */
/* Astronomical twilight starts/ends when the Sun's center is 18 degrees   */
/* below the horizon.                                                      */
```

```
#define day_astronomical_twilight_length(jd, lon, lat)  \
        __daylen__( jd, lon, lat, -18.0, 0 );
// #define day_astronomical_twilight_length(year,month,day,lon,lat)  \
//         __daylen__( year, month, day, lon, lat, -18.0, 0 )

/* This macro computes times for sunrise/sunset.                      */
/* Sunrise/set is considered to occur when the Sun's upper limb is    */
/* 35 arc minutes below the horizon (this accounts for the refraction */
/* of the Earth's atmosphere).                                        */
#define sun_rise_set(jd, lon, lat, rise, set)  \
        __sunriset__( jd, lon, lat, -50.0/60.0, 0, rise, set );
// #define sun_rise_set(year,month,day,lon,lat,rise,set)  \
//         __sunriset__( year, month, day, lon, lat, -50.0/60.0, 0, rise, set )

/* This macro computes the start and end times of civil twilight.     */
/* Civil twilight starts/ends when the Sun's center is 6 degrees below */
/* the horizon.                                                       */
#define civil_twilight(jd, lon, lat, start, end)  \
        __sunriset__( jd, lon, lat, -6.0, 0, start, end );
// #define civil_twilight(year,month,day,lon,lat,start,end)  \
//         __sunriset__( year, month, day, lon, lat, -6.0, 0, start, end )

/* This macro computes the start and end times of nautical twilight.  */
/* Nautical twilight starts/ends when the Sun's center is 12 degrees  */
/* below the horizon.                                                 */
#define nautical_twilight(jd, lon, lat, start, end)  \
        __sunriset__( jd, lon, lat, -12.0, 0, start, end );
// #define nautical_twilight(year,month,day,lon,lat,start,end)  \
//         __sunriset__( year, month, day, lon, lat, -12.0, 0, start, end )

/* This macro computes the start and end times of astronomical twilight. */
/* Astronomical twilight starts/ends when the Sun's center is 18 degrees */
/* below the horizon.                                                 */
#define astronomical_twilight(jd, lon, lat, start, end)  \
        __sunriset__( jd, lon, lat, -18.0, 0, start, end );
// #define astronomical_twilight(year,month,day,lon,lat,start,end)  \
//         __sunriset__( year, month, day, lon, lat, -18.0, 0, start, end )

/* Function prototypes */

double __daylen__( double jd, double lon, double lat,
                   double altit, int upper_limb );
// double __daylen__( int year, int month, int day, double lon, double lat,
                   //~ double altit, int upper_limb );
```

```
int __sunriset__( double jd, double lon, double lat,
                  double altit, int upper_limb, double *rise, double *set );
// int __sunriset__( int year, int month, int day, double lon, double lat,
                  //~ double altit, int upper_limb, double *rise, double *set );

void sunpos( double d, double *lon, double *r );

void sun_RA_dec( double d, double *RA, double *dec, double *r );

double revolution( double x );

double rev180( double x );

double GMST0( double d );

/* A small test program */

void main(void)
{
  int year,month,day;
  double d;  /* Days since 2000 Jan 0.0 (negative before) */
  double jd = time(0) / 86400.0 + EJD - J2000;
  double lon, lat;
  double daylen, civlen, nautlen, astrlen;
  double rise, set, civ_start, civ_end, naut_start, naut_end,
         astr_start, astr_end;
  int    rs, civ, naut, astr;

  lat = 41.9475360;
  lon = -88.7430640;

  int iy, im, id;
  double fd;
  // iauJd2cal(jd + J2000, 0, &iy, &im, &id, &fd);
  printf("\n");
  // printf ("\tDate \t\t : %4d/%2.2d/%2.2d\n", iy, im, id );
  // printf ("\tFD \t\t\t : %f\n", fd );
  printf ("\tMJD \t\t\t : %f \n", jd);

  // printf ("Number of days since 1970 Jan 1st " \
  // "is %ld \n", seconds / 86400);

  // printf( "Longitude (+ is east) and latitude (+ is north) : " );
  // scanf( "%lf %lf", &lon, &lat );
```

```
// for(;;)
// {
//  printf( "Input date ( yyyy mm dd ) (ctrl-C exits): " );
//  scanf( "%d %d %d", &year, &month, &day );

//  year = 2015;
//  month = 06;
//  day = 6;

//  d = days_since_2000_Jan_0(year,month,day) - lon/360.0;
//  printf( "JD %6.6fh \n", d + 2451545.0 );

daylen  = day_length(jd, lon, lat);
civlen  = day_civil_twilight_length(jd, lon, lat);
nautlen = day_nautical_twilight_length(jd, lon, lat);
astrlen = day_astronomical_twilight_length(jd, lon, lat);

printf( "\tDay length \t\t :%5.2f hours\n", daylen );
// printf( "With civil twilight         %5.2f hours\n", civlen );
// printf( "With nautical twilight      %5.2f hours\n", nautlen );
// printf( "With astronomical twilight  %5.2f hours\n", astrlen );
// printf( "Length of twilight: civil   %5.2f hours\n",
//         (civlen-daylen)/2.0);
// printf( "                    nautical  %5.2f hours\n",
//         (nautlen-daylen)/2.0);
// printf( "               astronomical  %5.2f hours\n",
//         (astrlen-daylen)/2.0);
rs   = sun_rise_set( jd, lon, lat, &rise, &set );
// rs = sun_rise_set( year, month, day, lon, lat, &rise, &set );
civ  = civil_twilight( jd, lon, lat, &civ_start, &civ_end );
// civ  = civil_twilight( year, month, day, lon, lat,
//                        &civ_start, &civ_end );
naut = nautical_twilight( jd, lon, lat, &naut_start, &naut_end );
// naut = nautical_twilight( year, month, day, lon, lat,
//                           &naut_start, &naut_end );
astr = astronomical_twilight( jd, lon, lat, &astr_start, &astr_end );
// astr = astronomical_twilight( year, month, day, lon, lat,
//                               &astr_start, &astr_end );

printf( "\tSun at south \t\t : %2.0f:%2.0f UTC\n",
        floor((rise+set)/2.0),
        floor(fmod((rise+set)/2.0, 1.0) * 60));

switch( rs )
{
```

```
  case 0:
    printf( "\tSun rises \t\t : %2.0f:%2.0f UTC\n",
            floor(rise),
            floor(fmod(rise, 1.0 ) * 60));
    printf( "\tSun sets \t\t : %2.0f:%2.0f UTC\n",
            floor(set),
            floor(fmod(set, 1.0 ) * 60));
    break;
  case +1:
    printf( "Sun above horizon\n" );
    break;
  case -1:
    printf( "Sun below horizon\n" );
    break;
}

switch( civ )
{
  case 0:
    printf( "\tCivil twilight \t\t : starts %5.2fh UTC\n"
            "\t\t\t\t : ends %5.2fh UTC\n", civ_start, civ_end );
    break;
  case +1:
    printf( "Never darker than civil twilight\n" );
    break;
  case -1:
    printf( "Never as bright as civil twilight\n" );
    break;
}

switch( naut )
{
  case 0:
    printf( "\tNautical twilight \t : starts %5.2fh UTC\n"
            "\t\t\t\t : ends %5.2fh UTC\n", naut_start, naut_end );
    break;
  case +1:
    printf( "Never darker than nautical twilight\n" );
    break;
  case -1:
    printf( "Never as bright as nautical twilight\n" );
  break;
}

switch( astr )
```

```
  {
    case 0:
      printf( "\tAstronomical twilight \t : starts %5.2fh UTC\n"
              "\t\t\t\t : ends %5.2fh UTC\n", astr_start, astr_end );
      break;
    case +1:
      printf( "Never darker than astronomical twilight\n" );
      break;
    case -1:
      printf( "Never as bright as astronomical twilight\n" );
      break;
  }
  printf("\n");
}

/* The "workhorse" function for sun rise/set times */
int __sunriset__( double jd, double lon, double lat,
                  double altit, int upper_limb, double *trise, double *tset )
// int __sunriset__( int year, int month, int day, double lon, double lat,
//                   double altit, int upper_limb, double *trise, double *tset )
/*********************************************************************/
/* Note: year,month,date = calendar date, 1801-2099 only.             */
/*       Eastern longitude positive, Western longitude negative       */
/*       Northern latitude positive, Southern latitude negative       */
/*       The longitude value IS critical in this function!            */
/*       altit = the altitude which the Sun should cross              */
/*               Set to -35/60 degrees for rise/set, -6 degrees       */
/*               for civil, -12 degrees for nautical and -18          */
/*               degrees for astronomical twilight.                   */
/*         upper_limb: non-zero -> upper limb, zero -> center         */
/*               Set to non-zero (e.g. 1) when computing rise/set     */
/*               times, and to zero when computing start/end of       */
/*               twilight.                                            */
/*        *rise = where to store the rise time                        */
/*        *set  = where to store the set  time                        */
/*                Both times are relative to the specified altitude,  */
/*                and thus this function can be used to comupte       */
/*                various twilight times, as well as rise/set times   */
/* Return value:  0 = sun rises/sets this day, times stored at        */
/*                    *trise and *tset.                               */
/*               +1 = sun above the specified "horizon" 24 hours.     */
/*                    *trise set to time when the sun is at south,    */
/*                    minus 12 hours while *tset is set to the south  */
/*                    time plus 12 hours. "Day" length = 24 hours     */
/*               -1 = sun is below the specified "horizon" 24 hours   */
```

```
/*                    "Day" length = 0 hours, *trise and *tset are         */
/*                    both set to the time when the sun is at south.       */
/*                                                                         */
/***************************************************************************/
{
  double d,       /* Days since 2000 Jan 0.0 (negative before) */
         gmsad,   /* */
         majd,    /* */
         sRA,     /* Sun's Right Ascension */
         sdec,    /* Sun's declination */
         sr,      /* Solar distance, astronomical units */
         sradius, /* Sun's apparent radius */
         t,       /* Diurnal arc */
         tsouth,  /* Time when Sun is at south */
         sidtime; /* Local sidereal time */

  int rc = 0;     /* Return cde from function - usually 0 */

  /* Compute d of 12h local mean solar time */
  // majd = days_since_2000_Jan_0(year,month,day);
  // printf( "MAJD %6.6f \n", majd );
  // printf( "AJD %6.6f \n", majd + 2451545.0 );
  // printf( "JD %6.6f \n", majd + 2451545.0 + 0.5 );
  d =  jd + 0.5 - lon / 360.0;
  // printf( "mean solar transit JD %6.9f \n", d + 2451545.0 );

  /* Compute local sideral time of this moment */
  gmsad = GMST0(d);
  // printf( "GMSAD %6.6fh \n", gmsad );
  sidtime = revolution( gmsad + 180.0 + lon );
  // printf( "LMSAD %6.6fh \n", sidtime );
  // printf( "LMST %6.6fh \n", sidtime / 15.0 );

  /* Compute Sun's RA + Decl at this moment */
  sun_RA_dec( d, &sRA, &sdec, &sr );

  /* Compute time when Sun is at south - in hours UT */
  tsouth = 12.0 - rev180(sidtime - sRA) / 15.0;

  /* Compute the Sun's apparent radius, degrees */
  sradius = 0.2666 / sr;

  /* Do correction to upper limb, if necessary */
  if ( upper_limb )
    altit -= sradius;
```

```
    /* Compute the diurnal arc that the Sun traverses to reach */
    /* the specified altitide altit: */
  {
    double cost;
    cost = ( sind(altit) - sind(lat) * sind(sdec) ) /
           ( cosd(lat) * cosd(sdec) );
    if ( cost >= 1.0 )
      rc = -1, t = 0.0;       /* Sun always below altit */
    else if ( cost <= -1.0 )
      rc = +1, t = 12.0;      /* Sun always above altit */
    else
      t = acosd(cost)/15.0;   /* The diurnal arc, hours */
  }

  /* Store rise and set times - in hours UT */
  *trise = tsouth - t;
  *tset  = tsouth + t;

  return rc;
}  /* __sunriset__ */

/* The "workhorse" function */


// double __daylen__( int year, int month, int day, double lon, double lat,
//                    double altit, int upper_limb )
double __daylen__( double jd, double lon, double lat,
                   double altit, int upper_limb )
/*********************************************************************/
/* Note: year,month,date = calendar date, 1801-2099 only.             */
/*       Eastern longitude positive, Western longitude negative       */
/*       Northern latitude positive, Southern latitude negative       */
/*       The longitude value is not critical. Set it to the correct   */
/*       longitude if you're picky, otherwise set to to, say, 0.0     */
/*       The latitude however IS critical - be sure to get it correct */
/*       altit = the altitude which the Sun should cross              */
/*               Set to -35/60 degrees for rise/set, -6 degrees       */
/*               for civil, -12 degrees for nautical and -18          */
/*               degrees for astronomical twilight.                   */
/*         upper_limb: non-zero -> upper limb, zero -> center         */
/*               Set to non-zero (e.g. 1) when computing day length   */
/*               and to zero when computing day+twilight length.      */
/*********************************************************************/
{
```

```
double  d,          /* Days since 2000 Jan 0.0 (negative before) */
        obl_ecl,    /* Obliquity (inclination) of Earth's axis */
        sr,         /* Solar distance, astronomical units */
        slon,       /* True solar longitude */
        sin_sdecl,  /* Sine of Sun's declination */
        cos_sdecl,  /* Cosine of Sun's declination */
        sradius,    /* Sun's apparent radius */
        t;          /* Diurnal arc */

/* Compute d of 12h local mean solar time */
// d = days_since_2000_Jan_0(year,month,day) + 0.5 - lon/360.0;
/* Compute d of 12h local mean solar time */
d = jd -  + 0.5 - lon/360.0;

/* Compute obliquity of ecliptic (inclination of Earth's axis) */
obl_ecl = 23.4393 - 3.563E-7 * d;

/* Compute Sun's position */
sunpos( d, &slon, &sr );

/* Compute sine and cosine of Sun's declination */
sin_sdecl = sind(obl_ecl) * sind(slon);
cos_sdecl = sqrt( 1.0 - sin_sdecl * sin_sdecl );

/* Compute the Sun's apparent radius, degrees */
sradius = 0.2666 / sr;

/* Do correction to upper limb, if necessary */
if ( upper_limb )
  altit -= sradius;

  /* Compute the diurnal arc that the Sun traverses to reach */
  /* the specified altitide altit: */
{
  double cost;
  cost = ( sind(altit) - sind(lat) * sin_sdecl ) /
         ( cosd(lat) * cos_sdecl );
  if ( cost >= 1.0 )
    t = 0.0;                      /* Sun always below altit */
  else if ( cost <= -1.0 )
    t = 24.0;                     /* Sun always above altit */
  else
    t = (2.0/15.0) * acosd(cost); /* The diurnal arc, hours */
}
return t;
```

```
}  /* __daylen__ */


/* This function computes the Sun's position at any instant */

void sunpos( double d, double *lon, double *r )
/*************************************************/
/* Computes the Sun's ecliptic longitude and distance */
/* at an instant given in d, number of days since     */
/* 2000 Jan 0.0.  The Sun's ecliptic latitude is not  */
/* computed, since it's always very near 0.           */
/*************************************************/
{
  double M,         /* Mean anomaly of the Sun */
         w,         /* Mean longitude of perihelion */
                    /* Note: Sun's mean longitude = M + w */
         e,         /* Eccentricity of Earth's orbit */
         E,         /* Eccentric anomaly */
         x, y,      /* x, y coordinates in orbit */
         v;         /* True anomaly */

  /* Compute mean elements */
  // M = fmod( 356.0470 + 0.9856002585 * d, 360.0 );
  M = fmod( 357.52911 + 0.985600281725 * d +
             -4.20718412047e-09 * d * d    +
              1.03430001369e-12 * d * d * d, 360.0 );
  w = 282.9404 + 4.70935E-5 * d;
  e = 0.016709 - 1.151E-9 * d;

  /* Compute true longitude and radius vector */
  E = M + e * RADEG * sind(M) * ( 1.0 + e * cosd(M) );
  x = cosd(E) - e;
  y = sqrt( 1.0 - e*e ) * sind(E);
  *r = sqrt( x*x + y*y );           /* Solar distance */
  v = atan2d( y, x );               /* True anomaly */
  *lon = fmod(v + w, 360.0);        /* True solar longitude */

  // if ( *lon >= 360.0 )
  //   *lon -= 360.0;                  /* Make it 0..360 degrees */
}

void sun_RA_dec( double d, double *RA, double *dec, double *r )
{
  double lon, obl_ecl, x, y, z;
```

```
  /* Compute Sun's ecliptical coordinates */
  sunpos( d, &lon, r );

  /* Compute ecliptic rectangular coordinates (z=0) */
  x = *r * cosd(lon);
  y = *r * sind(lon);

  /* Compute obliquity of ecliptic (inclination of Earth's axis) */
  obl_ecl = 23.439291 - 3.563E-7 * d;

  /* Convert to equatorial rectangular coordinates - x is uchanged */
  z = y * sind(obl_ecl);
  y = y * cosd(obl_ecl);

  /* Convert to spherical coordinates */
  *RA = atan2d( y, x );
  *dec = atan2d( z, sqrt(x*x + y*y) );

}  /* sun_RA_dec */


/*****************************************************************/
/* This function reduces any angle to within the first revolution */
/* by subtracting or adding even multiples of 360.0 until the     */
/* result is >= 0.0 and < 360.0                                   */
/*****************************************************************/

#define INV360    ( 1.0 / 360.0 )

double revolution( double x )
/*****************************************/
/* Reduce angle to within 0..360 degrees */
/*****************************************/
{
  return( x - 360.0 * floor( x * INV360 ) );
}  /* revolution */

double rev180( double x )
/*********************************************/
/* Reduce angle to within +180..+180 degrees */
/*********************************************/
{
  return( x - 360.0 * floor( x * INV360 + 0.5 ) );
}  /* revolution */
```

```
/**********************************************************************/
/* This function computes GMST0, the Greenwhich Mean Sidereal Time   */
/* at 0h UT (i.e. the sidereal time at the Greenwhich meridian at    */
/* 0h UT).  GMST is then the sidereal time at Greenwich at any       */
/* time of the day.  I've generelized GMST0 as well, and define it   */
/* as:  GMST0 = GMST - UT  --  this allows GMST0 to be computed at   */
/* other times than 0h UT as well.  While this sounds somewhat       */
/* contradictory, it is very practical:  instead of computing        */
/* GMST like:                                                        */
/*                                                                   */
/*  GMST = (GMST0) + UT * (366.2422/365.2422)                        */
/*                                                                   */
/* where (GMST0) is the GMST last time UT was 0 hours, one simply    */
/* computes:                                                         */
/*                                                                   */
/*  GMST = GMST0 + UT                                                */
/*                                                                   */
/* where GMST0 is not the GMST "at 0h UT" but at the current moment! */
/* Defined in this way, GMST0 will increase with about 4 min a       */
/* day.  It also happens that GMST0 (in degrees, 1 hr = 15 degr)     */
/* is equal to the Sun's mean longitude plus/minus 180 degrees!      */
/* (if we neglect aberration, which amounts to 20 seconds of arc     */
/* or 1.33 seconds of time)                                          */
/*                                                                   */
/**********************************************************************/

double GMST0( double d )
{
  double sidtim0;
  /* Sidtime at 0h UT = L (Sun's mean longitude) + 180.0 degr  */
  /* L = M + w, as defined in sunpos().  Since I'm too lazy to */
  /* add these numbers, I'll let the C compiler do it for me.  */
  /* Any decent C compiler will add the constants at compile   */
  /* time, imposing no runtime or code overhead.               */
  sidtim0 = revolution( ( 180.0 + 356.0470 +  282.9404) +
                        ( 0.9856002585 + 4.70935E-5 ) * d );
  sidtim0 = fmod( ( 180.0 + 356.0470 +  282.9404) +
                  ( 0.9856002585 + 4.70935E-5 ) * d, 360.0 );
  sidtim0 = fmod( ( 180 + 357.52911 + 282.9404 ) +
                  ( 0.985600281725 + 4.70935E-5 ) * d, 360.0);
  // sidtim0 = fmod( 280.4664567                +
  //                 d * ( 0.9856473601         +
  //                 d * ( 8.30124024641e-09    +
  //                 d * ( -5.48326848477e-11   +
```

```
  //                 d * ( -1.78956192374e-10   +
  //                 d * ( -11.37718852471e-12 ) ) ) ) ), 360.0 );
  return sidtim0;
}  /* GMST0 */
```