Package ‘SunCalcMeeus’

January 9, 2025

Type Package

Title Sun Position and Daylight Calculations

Version 0.1.2

Date 2025-01-09

Description Compute the position of the sun, and local solar time using Meeus' formulae. Compute day and/or night length using different twilight definitions or arbitrary sun elevation angles. This package is part of the 'r4photobiology' suite, Aphalo, P. J. (2015) <doi:10.19232/uv4pb.2015.1.14>. Algorithms from Meeus (1998, ISBN:0943396611).

License GPL (>= 2)

Depends R (>= 4.0.0)

Imports stats, tibble (>= 3.1.6), lubridate (>= 1.9.3), dplyr (>= 1.0.9)

Suggests knitr (>= 1.41), rmarkdown (>= 2.18), testthat (>= 3.2.1), roxygen2 (>= 7.3.0), lutz (>= 0.3.2), covr, spelling

LazyLoad yes

ByteCompile true

URL `https://docs.r4photobiology.info/SunCalcMeeus/`, `https://github.com/aphalo/SunCalcMeeus`

BugReports `https://github.com/aphalo/SunCalcMeeus/issues`

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Language en-US

Config/Needs/website rmarkdown

NeedsCompilation no

Author Pedro J. Aphalo [aut, cre] (<`https://orcid.org/0000-0003-3385-972X`>)

Maintainer Pedro J. Aphalo `<pedro.aphalo@helsinki.fi>`

Repository CRAN

Date/Publication 2025-01-09 18:30:07 UTC

Contents

SunCalcMeeus-package 2
as.solar_date 3
as_tod 4
day_night 5
format.solar_time 9
format.tod_time 9
irrad_extraterrestrial 10
is.solar_time 11
print.solar_time 12
print.tod_time 12
relative_AM 13
solar_time 15
sun_angles 17
tz_time_diff 19
validate_geocode 20

Index **22**

`SunCalcMeeus-package` *SunCalcMeeus: Sun Position and Daylight Calculations*

Description

Compute the position of the sun, and local solar time using Meeus' formulae. Compute day and/or night length using different twilight definitions or arbitrary sun elevation angles. This package is part of the 'r4photobiology' suite, Aphalo, P. J. (2015) doi:10.19232/uv4pb.2015.1.14. Algorithms from Meeus (1998, ISBN:0943396611).

Details

Please see the vignette *0: The R for Photobiology Suite* for a description of the suite.

Author(s)

Maintainer: Pedro J. Aphalo <`pedro.aphalo@helsinki.fi`> (ORCID)

References

Aphalo, Pedro J. (2015) The r4photobiology suite. *UV4Plants Bulletin*, 2015:1, 21-29. doi:10.19232/uv4pb.2015.1.14.

See Also

Useful links:

- `https://docs.r4photobiology.info/SunCalcMeeus/`
- `https://github.com/aphalo/SunCalcMeeus`
- Report bugs at `https://github.com/aphalo/SunCalcMeeus/issues`

Examples

```
# daylength
sunrise_time(lubridate::today(tzone = "EET"), tz = "EET",
             geocode = data.frame(lat = 60, lon = 25),
             unit.out = "hour")
day_length(lubridate::today(tzone = "EET"), tz = "EET",
           geocode = data.frame(lat = 60, lon = 25),
           unit.out = "hour")
sun_angles(lubridate::now(tzone = "EET"), tz = "EET",
           geocode = data.frame(lat = 60, lon = 25))
```

`as.solar_date` *Convert a solar_time object into solar_date object*

Description

Convert a solar_time object into solar_date object

Usage

```
as.solar_date(x, time)
```

Arguments

`x`	solar_time object.
`time`	an R date time object that provides the date part.

Details

Objects of class "solar_time" lack date information, it describes the time since local astronomical or true midnight. This function adds the date information from the argument passed to time `time` assembling a modified `time` object of class "solar_date".

Value

An object of class "solar.date" object derived from POSIXct. This is needed only for unambiguous formatting and printing.

See Also

Other Local solar time functions: `is.solar_time()`, `print.solar_time()`, `solar_time()`

as_tod *Convert datetime to time-of-day*

Description

Convert a datetime into a time of day expressed in hours, minutes or seconds from midnight in local time for a time zone. This conversion is useful when time-series data for different days needs to be compared or plotted based on the local time-of-day.

Usage

```
as_tod(x, unit.out = "hours", tz = NULL)
```

Arguments

`x`	a datetime object accepted by lubridate functions.
`unit.out`	character string, One of "tod_time", "hours", "minutes", or "seconds".
`tz`	character string indicating time zone to be used in output.

Value

A numeric vector of the same length as `x`. If `unit.out = "tod_time"` an object of class `"tod_time"` which a numeric vector as with `unit.out = "hours"` but with the class attribute set to `"tod_time"`, which dispatches to special `format()` and `print()` methods.

See Also

`solar_time`

Other Time of day functions: `format.tod_time()`, `print.tod_time()`

Examples

```
library(lubridate)
my_instants <- ymd_hms("2020-05-17 12:05:03") + days(c(0, 30))
my_instants
as_tod(my_instants)
as_tod(my_instants, unit.out = "tod_time")
```

day_night *Times for sun positions*

Description

Functions for calculating the timing of solar positions, given geographical coordinates and dates. They can be also used to find the time for an arbitrary solar elevation between 90 and -90 degrees by supplying "twilight" angle(s) as argument.

Usage

```
day_night(
  date = lubridate::now(tzone = "UTC"),
  tz = ifelse(lubridate::is.Date(date), "UTC", lubridate::tz(date)),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "none",
  unit.out = "hours"
)

day_night_fast(date, tz, geocode, twilight, unit.out)

is_daytime(
  date = lubridate::now(tzone = "UTC"),
  tz = ifelse(lubridate::is.Date(date), "UTC", lubridate::tz(date)),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "none",
  unit.out = "hours"
)

noon_time(
  date = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "none",
  unit.out = "datetime"
)

sunrise_time(
  date = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "datetime"
)

sunset_time(
  date = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(date),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "datetime"
)

day_length(
  date = lubridate::now(tzone = "UTC"),
  tz = "UTC",
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "hours"
)

night_length(
  date = lubridate::now(tzone = "UTC"),
  tz = "UTC",
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  twilight = "sunlight",
  unit.out = "hours"
)
```

Arguments

`date`	"vector" of `POSIXct` times or`Date` objects, any valid TZ is allowed, default is current date at Greenwich matching the default for `geocode`.
`tz`	character vector indicating time zone to be used in output and to interpret `Date` values passed as argument to `date`.
`geocode`	data frame with one or more rows and variables lon and lat as numeric values (degrees). If present, address will be copied to the output.
`twilight`	character string, one of "none", "rim", "refraction", "sunlight", "civil", "nautical", "astronomical", or a `numeric` vector of length one, or two, giving solar elevation angle(s) in degrees (negative if below the horizon).
`unit.out`	character string, One of "datetime", "day", "hour", "minute", or "second".

Details

Twilight names are interpreted as follows. "none": solar elevation = 0 degrees. "rim": upper rim of solar disk at the horizon or solar elevation = -0.53 / 2. "refraction": solar elevation = 0 degrees + refraction correction. "sunlight": upper rim of solar disk corrected for refraction, which is close to the value used by the online NOAA Solar Calculator. "civil": -6 degrees, "naval": -12 degrees, and "astronomical": -18 degrees. Unit names for output are as follows: "day", "hours", "minutes" and "seconds" times for sunrise and sunset are returned as times-of-day since midnight expressed in the chosen unit. "date" or "datetime" return the same times as datetime objects with TZ set (this is much slower than "hours"). Day length and night length are returned as numeric values expressed in hours when ‘"datetime"’ is passed as argument to `unit.out`. If twilight is a numeric vector of length two, the element with index 1 is used for sunrise and that with index 2 for sunset.

`is_daytime()` supports twilight specifications by name, a test like `sun_elevation() > 0` may be used directly for a numeric angle.

Value

A tibble with variables day, tz, twilight.rise, twilight.set, longitude, latitude, address, sunrise, noon, sunset, daylength, nightlength or the corresponding individual vectors.

The value returned represents an instant in time or a duration. The class of the object returned varies depending on the argument passed to parameter `unit.out`. If `unit.out = "datetime"`, the returned value is a "POSIXct" vector, otherwise it is a "numeric" vector.

`is_daytime()` returns a logical vector, with `TRUE` for day time and `FALSE` for night time.

`noon_time`, `sunrise_time` and `sunset_time` return a vector of POSIXct times

`day_length` and `night_length` return numeric a vector giving the length in hours

Warning

Be aware that R's `Date` class does not save time zone metadata. This can lead to ambiguities in the current implementation based on time instants. The argument passed to `date` should be of class `POSIXct`, in other words an instant in time, from which the correct date will be computed based on the `tz` argument.

The time zone in which times passed to `date` as argument are expressed does not need to be the local one or match the geocode, however, the returned values will be in the same time zone as the input.

Note

Function `day_night()` is an implementation of Meeus equations as used in NOAAs on-line web calculator, which are very precise and valid for a very broad range of dates. For sunrise and sunset the times are affected by refraction in the atmosphere, which does in turn depend on weather conditions. The effect of refraction on the apparent position of the sun is only an estimate based on "typical" conditions. The more tangential to the horizon is the path of the sun, the larger the effect of refraction is on the times of visual occlusion of the sun behind the horizon—i.e. the largest timing errors occur at high latitudes. The computation is not defined for latitudes 90 and -90 degrees, i.e. at the poles.

There exists a different R implementation of the same algorithms called "AstroCalcPureR" available as function `astrocalc4r` in package 'fishmethods'. Although the equations used are almost all the same, the function signatures and which values are returned differ. In particular, the implementation in 'photobiology' splits the calculation into two separate functions, one returning angles at given instants in time, and a separate one returning the timing of events for given dates. In 'fishmethods' (= 1.11-0) there is a bug in function astrocalc4r() that affects sunrise and sunset times. The times returned by the functions in package 'photobiology' have been validated against the NOAA base implementation.

In the current implementation functions `sunrise_time`, `noon_time`, `sunset_time`, `day_length`, `night_length` and `is_daytime` are all wrappers on `day_night`, so if more than one quantity is needed it is preferable to directly call `day_night` and extract the different components from the returned list.

`night_length` returns the length of night-time conditions in one day (00:00:00 to 23:59:59), rather than the length of the night between two consecutive days.

References

The primary source for the algorithm used is the book: Meeus, J. (1998) Astronomical Algorithms, 2 ed., Willmann-Bell, Richmond, VA, USA. ISBN 978-0943396613.

A different implementation is available at https://github.com/NEFSC/READ-PDB-AstroCalc4R/ and in R paclage 'fishmethods'. In 'fishmethods' (= 1.11-0) there is a bug in function astrocalc4r() that affects sunrise and sunset times.

An interactive web page using the same algorithms is available at https://gml.noaa.gov/grad/solcalc/. There are small differences in the returned times compared to our function that seem to be related to the estimation of atmospheric refraction (about 0.1 degrees).

See Also

`sun_angles`.

Other astronomy related functions: `format.solar_time()`, `sun_angles()`

Examples

```
library(lubridate)

my.geocode <- data.frame(lon = 24.93838,
                         lat = 60.16986,
                         address = "Helsinki, Finland")

day_night(ymd("2015-05-30", tz = "EET"),
          geocode = my.geocode)
day_night(ymd("2015-05-30", tz = "EET") + days(1:10),
          geocode = my.geocode,
          twilight = "civil")
sunrise_time(ymd("2015-05-30", tz = "EET"),
             geocode = my.geocode)
noon_time(ymd("2015-05-30", tz = "EET"),
          geocode = my.geocode)
sunset_time(ymd("2015-05-30", tz = "EET"),
            geocode = my.geocode)
day_length(ymd("2015-05-30", tz = "EET"),
           geocode = my.geocode)
day_length(ymd("2015-05-30", tz = "EET"),
           geocode = my.geocode,
           unit.out = "day")
is_daytime(ymd("2015-05-30", tz = "EET") + hours(c(0, 6, 12, 18, 24)),
           geocode = my.geocode)
is_daytime(ymd_hms("2015-05-30 03:00:00", tz = "EET"),
           geocode = my.geocode)
is_daytime(ymd_hms("2015-05-30 00:00:00", tz = "UTC"),
           geocode = my.geocode)
is_daytime(ymd_hms("2015-05-30 03:00:00", tz = "EET"),
           geocode = my.geocode,
           twilight = "civil")
is_daytime(ymd_hms("2015-05-30 00:00:00", tz = "UTC"),
           geocode = my.geocode,
           twilight = "civil")
```

`format.solar_time` *Encode in a Common Format*

Description

Format a `solar_time` object for pretty printing

Usage

```
## S3 method for class 'solar_time'
format(x, ..., sep = ":")
```

Arguments

`x`	an R object
`...`	ignored
`sep`	character used as separator

Value

A character string with the time formatted as "HH:MM:SS", where ":" is the argument passed to `sep`.

See Also

Other astronomy related functions: `day_night()`, `sun_angles()`

`format.tod_time` *Encode in a Common Format*

Description

Format a `tod_time` object into a character string for pretty printing.

Usage

```
## S3 method for class 'tod_time'
format(x, ..., sep = ":")
```

Arguments

`x`	an R object
`...`	ignored
`sep`	character used as separator

Value

A character string with the time formatted as "HH:MM:SS", where ":" is the argument passed to `sep`.

See Also

Other Time of day functions: `as_tod()`, `print.tod_time()`

`irrad_extraterrestrial` *Extraterrestrial solar irradiance*

Description

Estimate of down-welling solar (short wave) irradiance at the top of the atmosphere above a location on Earth, computed based on angles, Sun-Earth distance and the solar constant. Astronomical computations are done with function `sun_angles()`.

Usage

```
irrad_extraterrestrial(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = data.frame(lon = 0, lat = 51.5, address = "Greenwich"),
  solar.constant = "NASA"
)
```

Arguments

`time`	A "vector" of POSIXct Time, with any valid time zone (TZ) is allowed, default is current time.
`tz`	character string indicating time zone to be used in output.
`geocode`	data frame with variables lon and lat as numeric values (degrees), nrow > 1, allowed.
`solar.constant`	numeric or character If character, "WMO" or "NASA", if numeric, an irradiance value in the same units as the value to be returned.

Value

Numeric vector of extraterrestrial irradiance (in W / m2 if solar constant is a character value).

See Also

Function `sun_angles`.

Examples

```
library(lubridate)

irrad_extraterrestrial(ymd_hm("2021-06-21 12:00", tz = "UTC"))

irrad_extraterrestrial(ymd_hm("2021-12-21 20:00", tz = "UTC"))

irrad_extraterrestrial(ymd_hm("2021-06-21 00:00", tz = "UTC") + hours(1:23))
```

`is.solar_time` *Query class*

Description

Query class

Usage

```
is.solar_time(x)

is.solar_date(x)
```

Arguments

`x`	an R object.

Value

A logical value indicating if the object `x` is of class `"solar_time"` or `"solar_date"`, depending on the function.

See Also

Other Local solar time functions: `as.solar_date()`, `print.solar_time()`, `solar_time()`

`print.solar_time` *Print solar time and solar date objects*

Description

The object `x` is printed and returned invisibly.

Usage

```
## S3 method for class 'solar_time'
print(x, ...)

## S3 method for class 'solar_date'
print(x, ...)
```

Arguments

`x`	an R object
`...`	passed to `format` method

Value

Returns object `x`, invisibly.

Note

Default is to print the underlying POSIXct or Date as a solar time.

See Also

Other Local solar time functions: `as.solar_date()`, `is.solar_time()`, `solar_time()`

`print.tod_time` *Print time-of-day objects*

Description

Defaults to print the underlying `numeric` vector as a solar time.

Usage

```
## S3 method for class 'tod_time'
print(x, ...)
```

Arguments

`x`	an R object
`...`	passed to `format` method

Value

Returns object `x`, invisibly.

See Also

Other Time of day functions: `as_tod()`, `format.tod_time()`

`relative_AM` *Relative Air Mass (AM)*

Description

Approximate relative air mass (AM) computed from the sun's apparent or true position (sun elevation or sun zenith angle) or from geographic and time coordinates.

Usage

```
relative_AM(
  elevation.angle = NULL,
  zenith.angle = NULL,
  occluded.value = NA_real_
)

relative_AMt(
  elevation.angle = NULL,
  zenith.angle = NULL,
  occluded.value = NA_real_
)

relative_AM_geotime(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  occluded.value = NA_real_
)

relative_AMt_geotime(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  occluded.value = NA_real_
)
```

Arguments

`elevation.angle`, `zenith.angle`	numeric vector Angle in degrees for the sun position. An argument should be passed to one and only one of `elevation_angle` or `zenith_angle`.
`occluded.value`	numeric Value to return when elevation angle is negative (sun below the horizon).
`time`	A "vector" of POSIXct Time, with any valid time zone (TZ) is allowed, default is current time.
`tz`	character string indicating time zone to be used in output.
`geocode`	data frame with variables lon and lat as numeric values (degrees), nrow > 1, allowed.

Details

Function `relative_AM()` implements equation (3) in Kasten and Young (1989). This equation is only an approximation to the tabulated values in the same paper and based on the apparent position of the sun as observed from Earth surface. `relative_AMt()` implements equation (5) in Young (1994). This equation is only an approximation to the tabulated values based on the true or astronomical position of the sun.

In both cases returned values are rounded to three significant digits.

Function `relative_AM_geotime()` is a wrapper on `relative_AM()` that calls function `sun_elevation()` to obtain the apparent position of the sun from the geographic and time coordinates. Function `relative_AMt_geotime()` is a wrapper on `relative_AMt()` that calls function `sun_elevation()` to obtain the true position of the sun from the geographic and time coordinates. At very low sun elevations the values returned by these two functions differ slightly because of the use of different approximations to correct for atmospheric refraction.

Value

A numeric vector with the relative air mass values.

Note

Although relative air mass is not defined when the sun is not visible, returning a value different from the default `NA` might be useful in some cases and made possible by passing an argument to parameter `occluded.value`.

References

F. Kasten, A. T. Young (1989) Revised optical air mass tables and approximation formula. Applied Optics, 28, 4735-4738. doi:10.1364/AO.28.004735.

Young, A. T. (1994) Air mass and refraction. Applied Optics, 33, 1108-1110. doi:10.1364/AO.33.001108

See Also

`sun_angles`

Examples

```
# using the apparent sun elevation
relative_AM(elevation.angle = c(90, 60, 30, 1, -10))
relative_AM(elevation.angle = c(90, 60, 30, 1, -10),
            occluded.value = Inf)
relative_AM(zenith.angle = 0)

# using the true or astronomical sun elevation
relative_AMt(elevation.angle = c(90, 60, 30, 1, -10))
relative_AMt(elevation.angle = c(90, 60, 30, 1, -10),
            occluded.value = Inf)
relative_AMt(zenith.angle = 0)

# Define example geographic and time coordinates
baires.geo <-
  data.frame(lat = 34.60361, lon = -58.38139, address = "Buenos Aires")

# using time and geographic coordinates
library(lubridate)
relative_AM_geotime(ymd_hms("2014-06-23 12:00:00",
                            tz = "America/Argentina/Buenos_Aires"),
                    geocode = baires.geo)
relative_AMt_geotime(ymd_hms("2014-06-23 12:00:00",
                             tz = "America/Argentina/Buenos_Aires"),
                    geocode = baires.geo)
relative_AM_geotime(ymd_hms("2014-06-23 12:00:00",
                            tz = "America/Argentina/Buenos_Aires") +
                      hours(0:12),
                    geocode = baires.geo)
```

solar_time *Local solar time*

Description

`solar_time()` computes the time of day expressed in seconds since the astronomical midnight using and instant in time and a geocode as input. Solar time is useful when we want to plot data according to the local solar time rather than the local time in use at a time zone. How the returned instant in time is expressed depends on the argument passed to `unit.out`.

Usage

```
solar_time(
  time = lubridate::now(),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  unit.out = "time"
)
```

Arguments

`time`	POSIXct Time, any valid time zone (TZ) is allowed, default is current time.
`geocode`	data frame with variables lon and lat as numeric values (degrees).
`unit.out`	character string, One of "datetime", "time", "hour", "minute", or "second".

Details

Solar time is determined by the position of the sun in the sky and it almost always differs from the time expressed in the local time coordinates in use. The differences can vary from a few minutes up to a couple of hours depending on the exact location within the time zone and the use or not of daylight saving time.

Value

In all cases solar time is expressed as time since local astronomical midnight and, thus, lacks date information. If `unit.out = "time"`, a numeric value in seconds with an additional class attribute "solar_time"; if `unit.out = "datetime"`, a "POSIXct" value in seconds from midnight but with an additional class attribute "solar_date"; if `unit.out = "hour"` or `unit.out = "minute"` or `unit.out = "second"`, a numeric value.

Warning!

Returned values are computed based on the time zone of the argument for parameter time. In the case of solar time, this timezone does not affect the result. However, in the case of solar dates the date part may be off by one day, if the time zone does not match the coordinates of the geocode value provided as argument.

Note

The algorithm is approximate, it calculates the difference between local solar noon and noon in the time zone of `time` and uses this value for the whole day when converting times into solar time. Days are not exactly 24 h long. Between successive days the shift is only a few seconds, and this leads to a small jump at midnight.

See Also

`as_tod`

Other Local solar time functions: `as.solar_date()`, `is.solar_time()`, `print.solar_time()`

Examples

```
BA.geocode <-
  data.frame(lon = -58.38156, lat = -34.60368, address = "Buenos Aires, Argentina")
sol_t <- solar_time(lubridate::dmy_hms("21/06/2016 10:00:00", tz = "UTC"),
                    BA.geocode)
sol_t
class(sol_t)

sol_d <- solar_time(lubridate::dmy_hms("21/06/2016 10:00:00", tz = "UTC"),
                  BA.geocode,
                  unit.out = "datetime")
sol_d
class(sol_d)
```

`sun_angles` *Sun angles*

Description

Function `sun_angles()` returns the solar angles and Sun to Earth relative distance for given times and locations using a very accurate algorithm. Convenience functions `sun_azimuth()`, `sun_elevation()`, `sun_zenith_angle()` and `distance_to_sun()` are wrappers on `sun_angles()` that return individual vectors.

Usage

```
sun_angles(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_angles_fast(time, tz, geocode, use.refraction)

sun_elevation(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_zenith_angle(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_azimuth(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)
```

```
distance_to_sun(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)
```

Arguments

`time`	A "vector" of POSIXct Time, with any valid time zone (TZ) is allowed, default is current time.
`tz`	character string indicating time zone to be used in output.
`geocode`	data frame with variables lon and lat as numeric values (degrees), nrow > 1, allowed.
`use.refraction`	logical Flag indicating whether to correct for fraction in the atmosphere.

Details

This function is an implementation of Meeus equations as used in NOAA's on-line web calculator, which are precise and valid for a very broad range of dates (years -1000 to 3000 at least). The apparent solar elevations near sunrise and sunset are affected by refraction in the atmosphere, which does in turn depend on weather conditions. The effect of refraction on the apparent position of the sun is only an estimate based on "typical" conditions for the atmosphere. The computation is not defined for latitudes 90 and -90 degrees, i.e. exactly at the poles. The function is vectorized and in particular passing a vector of times for a single geocode enhances performance very much as the equation of time, the most time consuming step, is computed only once.

For improved performance, if more than one angle is needed it is preferable to directly call `sun_angles` instead of the wrapper functions as this avoids the unnecesary recalculation.

Value

A `data.frame` with variables `time` (in same TZ as input), `TZ`, `solartime`, `longitude`, `latitude`, `address`, `azimuth`, `elevation`, `declination`, `eq.of.time`, `hour.angle`, and `distance`. If a data frame with multiple rows is passed to `geocode` and a vector of times longer than one is passed to `time`, sun position for all combinations of locations and times are returned by `sun_angles`. Angles are expressed in degrees, `solartime` is a vector of class `"solar.time"`, `distance` is expressed in relative sun units.

Important!

Given an instant in time and a time zone, the date is computed from these, and may differ by one day to that at the location pointed by `geocode` at the same instant in time, unless the argument passed to `tz` matches the time zone at this location.

Note

There exists a different R implementation of the same algorithms called "AstroCalcPureR" available as function `astrocalc4r` in package 'fishmethods'. Although the equations used are almost all the same, the function signatures and which values are returned differ. In particular, the present implementation splits the calculation into two separate functions, one returning angles at given instants in time, and a separate one returning the timing of events for given dates.

References

The primary source for the algorithm used is the book: Meeus, J. (1998) Astronomical Algorithms, 2 ed., Willmann-Bell, Richmond, VA, USA. ISBN 978-0943396613.

A different implementation is available at `https://github.com/NEFSC/READ-PDB-AstroCalc4R/`.

An interactive web page using the same algorithms is available at `https://gml.noaa.gov/grad/solcalc/`. There are small differences in the returned times compared to our function that seem to be related to the estimation of atmospheric refraction (about 0.1 degrees).

See Also

Other astronomy related functions: `day_night()`, `format.solar_time()`

Examples

```
library(lubridate)
sun_angles()
sun_azimuth()
sun_elevation()
sun_zenith_angle()
sun_angles(ymd_hms("2014-09-23 12:00:00"))
sun_angles(ymd_hms("2014-09-23 12:00:00"),
           geocode = data.frame(lat=60, lon=0))
sun_angles(ymd_hms("2014-09-23 12:00:00") + minutes((0:6) * 10))
```

`tz_time_diff` *Time difference between two time zones*

Description

Returns the difference in local time expressed in hours between two time zones at a given instant in time. The difference due to daylight saving time or Summer and Winter time as well as historical changes in time zones are taken into account.

Usage

```
tz_time_diff(
  when = lubridate::now(),
  tz.target = lubridate::tz(when),
  tz.reference = "UTC"
)
```

Arguments

`when`	datetime A time instant
`tz.target, tz.reference`	character Two time zones using names recognized by functions from package 'lubridate'

Value

A `numeric` value.

Note

This function is implemented using functions from package 'lubridate'. For details on the handling of time zones, please, consult the documentation for `Sys.timezone` about system differences in time zone names and handling.

`validate_geocode` *Validate a geocode*

Description

Test validity of a geocode or ensure that a geocode is valid.

Usage

```
validate_geocode(geocode)

is_valid_geocode(geocode)

length_geocode(geocode)

na_geocode()
```

Arguments

`geocode`	data.frame with geocode data in columns `"lat"`, `"lon"`, and possibly also `"address"`.

Details

`validate_geocode` Converts to tibble, checks data bounds, converts address to character if it is not already a character vector, or add character NAs if the address column is missing.

`is_valid_geocode` Checks if a geocode is valid, returning 0L if not, and the number of row otherwise.

Value

A valid geocode stored in a tibble.

FALSE for invalid, TRUE for valid.

FALSE for invalid, number of rows for valid.

A geo_code tibble with all fields set to suitable NAs.

Examples

```
validate_geocode(NA)
validate_geocode(data.frame(lon = -25, lat = 66))

is_valid_geocode(NA)
is_valid_geocode(1L)
is_valid_geocode(data.frame(lon = -25, lat = 66))

na_geocode()
```

Index

∗ **Local solar time functions**
- as.solar_date, 3
- is.solar_time, 11
- print.solar_time, 12
- solar_time, 15

∗ **Time of day functions**
- as_tod, 4
- format.tod_time, 9
- print.tod_time, 12

∗ **astronomy related functions**
- day_night, 5
- format.solar_time, 9
- sun_angles, 17

as.solar_date, 3, *11*, *12*, *16*
as_tod, 4, *10*, *13*, *16*

day_length *(*day_night*)*, 5
day_night, 5, *9*, *19*
day_night_fast *(*day_night*)*, 5
distance_to_sun *(*sun_angles*)*, 17

format.solar_time, *8*, 9, *19*
format.tod_time, *4*, 9, *13*

irrad_extraterrestrial, 10
is.solar_date *(*is.solar_time*)*, 11
is.solar_time, *3*, 11, *12*, *16*
is_daytime *(*day_night*)*, 5
is_valid_geocode *(*validate_geocode*)*, 20

length_geocode *(*validate_geocode*)*, 20

na_geocode *(*validate_geocode*)*, 20
night_length *(*day_night*)*, 5
noon_time *(*day_night*)*, 5

print.solar_date *(*print.solar_time*)*, 12
print.solar_time, *3*, *11*, 12, *16*
print.tod_time, *4*, *10*, 12

relative_AM, 13
relative_AM_geotime *(*relative_AM*)*, 13
relative_AMt *(*relative_AM*)*, 13
relative_AMt_geotime *(*relative_AM*)*, 13

solar_time, *3*, *4*, *11*, *12*, 15
sun_angles, *8*, *9*, *11*, *14*, 17
sun_angles_fast *(*sun_angles*)*, 17
sun_azimuth *(*sun_angles*)*, 17
sun_elevation *(*sun_angles*)*, 17
sun_zenith_angle *(*sun_angles*)*, 17
SunCalcMeeus *(*SunCalcMeeus-package*)*, 2
SunCalcMeeus-package, 2
sunrise_time *(*day_night*)*, 5
sunset_time *(*day_night*)*, 5
Sys.timezone, *20*

tz_time_diff, 19

validate_geocode, 20