VedicDateTime: An R package to implement Vedic calendar system

Contents

Introduction

The whole concept of *VedicDateTime* is based on the Hindu calendar system. The lunisolar Hindu calendar and its aspects are as old as *Surya Siddhanta*, an ancient Sanskrit text on astronomy [1]. Even today, this calendar is prevalent in India and it is mostly found in *Hindu Panchanga* an Indian almanac that details information about the various festivals, rituals, and planetary combinations and has been in use since time immemorial in the Indian subcontinent. This Panchanga can be commonly found in most Indian homes[2].

The *VedicDateTime* package aims to convert the commonly used Gregorian date-time to the Vedic date-time as per the calculations of the *Panchang*. Panchanga which means 'five arms' consists of the five most important parts - *Tithi, Vaara, Nakshatra, Yoga,* and *Karana*. The *Vaara* or the day of the week is based on Sun alone; *Tithi* and *Karana* are based upon Moon alone; *Nakshatra* and *Yoga* are based upon both Moon and Sun. This makes the Hindu calendar a true lunisolar calendar. *VedicDateTime* contributes a new calendar system in R that can have huge potential to discover meaningful patterns in natural time series.

Before we get into calendar systems, we need to be clear with the concept of *Ayanamsha*. The Earth revolves anti-clockwise around the Sun in an elliptical orbit. The time taken by the earth to arrive at one vernal equinox from the other vernal equinox after one revolution is called as **tropical year**. At the end of the tropical year if we consider the position of the Earth with respect to a fixed star in the zodiac, the Earth appears to lie 50.26 seconds of celestial longitude to the west of its original position. In order to arrive at the same position with respect to a fixed star in the zodiac, the time taken is called as **sidereal year**. This is the result of precession of the Earth's axis due to its wobble in clockwise direction. The precession of the equinoxes increases by 50.2 seconds every year. The distance between the Vernal equinox and the 1st point of Aries (Mesha) on the fixed zodiac is progressively increasing. The distance at any given point of time is called as *Ayanamsha*.

We have two calendar systems– **Sayana** and **Nirayana**. *Sayana calendars*, also called as tropical calendars which means with *Ayana*. *Nirayana calendars*, also called as sidereal calendars which means without *Ayana*. The *Nirayana* values can be found out by subtracting the *Ayanamsa* from the *Sayana* values.

Each aspect (function) of the package is explained below. Each and every detail concerning each function is mentioned there and then.

```
library(VedicDateTime)
```

Panchanga

Tithi

Tithi can be called as a lunar day. When the moon traverses 12° in longitude relative to the sun, a tithi is completed. Fifteen tithis in the waxing phase (*Shukla paksha*) with end tithi as full moon day (*Purnima*) and other fifteen tithis in the waning phase (*Krishna paksha*) with end tithi as new moon day (*Amavasya*). These phases together constitute a lunar month which is of thirty tithis.

Tithi doesn't require *Ayanamsa* because it is the difference of longitudes and even if *Ayanamsa* is subtracted from each longitude, it doesn't matter[3].

[1] Oak, N., & Bhaty, R. (2019). Ancient updates to Surya-Siddhanta. Indiafacts.Org. Retrieved September 4, 2022, from https://indiafacts.org/ancient-updates-to-surya-siddhanta/

[2] Charak, K. S. (2012). Elements of Vedic Astrology (2 Volume Set) (Third Edition). UMA Publications.

[3] Ramakumar, K. L. (2011). Panchangam calculations. https://archive.org/details/PanchangamCalculations

Calculation of Tithi

```
tithi <- function(jd, place) {
    # Tithi as -> 1 = Shukla paksha prathama, 2 = Shukla
    # paksha dvitiya,...

    tz = place[3]  #Timezone of the place

    # 1. Find time of sunrise at a given place
    rise = sunrise(jd, place)[1] - (tz/24)

    # 2. Find tithi on this Julian day number at a given
    # place
    moon_phase = lunar_phase(rise)
    today = ceiling(moon_phase/12)
    degrees_left = today * 12 - moon_phase

    # 3. Compute longitudinal differences at intervals of
    # 0.25 days from sunrise
    offsets = c(0.25, 0.5, 0.75, 1)
    lunar_longitude_diff = c()
    solar_longitude_diff = c()
    relative_motion = c()
    for (i in 1:length(offsets)) {
        lunar_longitude_diff <- append(lunar_longitude_diff,
            ((moon_longitude(rise + offsets[i]) - moon_longitude(rise))%%360))
        solar_longitude_diff <- append(solar_longitude_diff,
            ((sun_longitude(rise + offsets[i]) - sun_longitude(rise))%%360))
        relative_motion <- append(relative_motion, (lunar_longitude_diff[i] -
            solar_longitude_diff[i]))
    }
    # 4. Find end time by 4-point inverse Lagrange
    # interpolation
    y = relative_motion
    x = offsets
    # Compute fraction of day (after sunrise) needed to
    # traverse 'degrees_left'
    approx_end = inverse_lagrange(x, y, degrees_left)
    ends = (rise + approx_end - jd) * 24 + tz
    answer = c(as.integer(today), to_dms(ends))

    # 5. Check for skipped tithi
    moon_phase_tom = lunar_phase(rise + 1)
    tomorrow = ceiling(moon_phase_tom/12)
    if (((tomorrow - today)%%30) > 1) {
        # interpolate again with same (x,y)
        leap_tithi = today + 1
        degrees_left = leap_tithi * 12 - moon_phase
        approx_end = inverse_lagrange(x, y, degrees_left)
        ends = (rise + approx_end - jd) * 24 + tz
        answer <- append(answer, c(as.integer(leap_tithi), to_dms(ends)))
    }
    return(answer)
}
```

At first, we take the **date** as Julian's day number and **place** as the latitude, longitude, and time zone of the place for which the calculations are to be made as parameters of the function. The **sunrise** for the given date and place is found using another function which uses the *Swiss-ephemeris* to get the sunrise as a Julian day number. Tithi is found out during the sunrise as the difference between lunar and solar longitudes divided by 12. Tithi is generally obtained as a decimal number. The fractional part of the Tithi is used to find out the remaining degrees to the next Tithi which is used to find the time when that Tithi ends. The time taken to complete the remaining degrees is given by *Inverse Lagrange's interpolation* of the Tithis found by using the lunar and solar longitudes at the intervals of 0.25 days from sunrise. The obtained Tithi and its ending time are added to the output vector. Sometimes two Tithis can occur on a given date so, we check for skipped tithi by checking the Tithi of the next day and if a Tithi is skipped then, the skipped Tithi is included in the output with its ending time.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
tithi(jd, place)
#> [1] 20 20 55 35
```

In the above output, the first number represents the number associated with the respective Tithi. 20 represents **Krishna paksha panchami**. This tithi ends at **20 hours 55 minutes and 35 seconds** which is represented by the rest of the numbers.

```
tithi(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] 18  6 11 26 19 26 59 58
```

In the above output, 6 numbers are returned out of which the first 3 numbers represent the Tithi name and ending time of the first Tithi and the next 3 numbers represent the Tithi name and ending time of the second Tithi. Here, on **17th June 2022**, two Tithis had occurred - 18 which represents **Krishna paksha trititya** ending at **6hrs 11mins and 26secs** and 19 which represents **Krishna paksha chaturthi** ending at **2hrs 59mins and 58secs** on the next day **18th June 2022**.

The timings greater than 24:00 represents the time past midnight. The ending time of that Tithi in the next day can be obtained by subtracting 24 from the hours.

Tithi's name

The Tithi number and ending date are obtained from the tithi function. The Tithi's name associated with each Tithi number is returned with its ending time by this function.

The following are **30 Tithis** with their corresponding numbers -

```
#> [1] "1  -  Shukla paksha prathama"
#> [1] "2  -  Shukla paksha dvitiya"
#> [1] "3  -  Shukla paksha trititya"
#> [1] "4  -  Shukla paksha chaturthi"
#> [1] "5  -  Shukla paksha panchami"
#> [1] "6  -  Shukla paksha sashti"
#> [1] "7  -  Shukla paksha saptami"
#> [1] "8  -  Shukla paksha ashtami"
#> [1] "9  -  Shukla paksha navami"
#> [1] "10  -  Shukla paksha dashmi"
#> [1] "11  -  Shukla paksha ekadashi"
#> [1] "12  -  Shukla paksha dvadashi"
#> [1] "13  -  Shukla paksha trayodashi"
#> [1] "14  -  Shukla paksha chaturdashi"
#> [1] "15  -  Poornima"
#> [1] "16  -  Krishna paksha prathama"
#> [1] "17  -  Krishna paksha dvitiya"
#> [1] "18  -  Krishna paksha trititya"
```

```
#> [1] "19  -  Krishna paksha chaturthi"
#> [1] "20  -  Krishna paksha panchami"
#> [1] "21  -  Krishna paksha sashti"
#> [1] "22  -  Krishna paksha saptami"
#> [1] "23  -  Krishna paksha ashtami"
#> [1] "24  -  Krishna paksha navami"
#> [1] "25  -  Krishna paksha dashmi"
#> [1] "26  -  Krishna paksha ekadashi"
#> [1] "27  -  Krishna paksha dvadashi"
#> [1] "28  -  Krishna paksha trayodashi"
#> [1] "29  -  Krishna paksha chaturdashi"
#> [1] "30  -  Amavasya"
```

Get name(s) of the Tithi for given Julian day number and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
get_tithi_name(jd, place)
#> [1] "Krishna paksha panchami till 20:55:35"
```

```
get_tithi_name(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))
#> [1] "Krishna paksha trititya till 6:11:26 & Krishna paksha chaturthi till 26:59:58"
```

Vaara

Vaara is the day of the week very similar to the Gregorian calendar.

Calculation of Vaara

```
vaara <- function(jd) {
    return(as.integer(ceiling(jd + 1)%%7) + 1)
}
```

Vaara for a given Julian day number

```
vaara(2459778)
#> [1] 1
```

```
vaara(gregorian_to_jd(6, 8, 2022))
#> [1] 7
```

Vaara's name

The following are **7 Vaaras** with their corresponding numbers -

```
#> [1] "1  -  Ravivar"
#> [1] "2  -  Somvar"
#> [1] "3  -  Mangalwar"
#> [1] "4  -  Budhwar"
#> [1] "5  -  Guruwar"
#> [1] "6  -  Shukrawar"
#> [1] "7  -  Shaniwar"
```

Get name of the Vaara for given Julian day number.

```
get_vaara_name(2459778)
#> [1] "Ravivara"
```

```
get_vaara_name(gregorian_to_jd(6, 8, 2022))
#> [1] "Shanivara"
```

Nakshatra

The zodiac is divided into **27 Nakshatras**. The absolute longitude of the Moon is used to calculate the Nakshatra of the day. This requires **Ayanamsa** to be subtracted from the moon's tropical (*Sayana*) longitude to get its sidereal (*Nirayana*) longitude which can be used to find out the nakshatra.

Calculation of Nakshatra

```
nakshatra <- function(jd, place) {
    # Nakshatra as -> 1 = Ashwini, 2 = Bharani, ..., 27 =
    # Revati

    # Set Lahiri ayanamsa
    swe_set_sid_mode(SE$SIDM_LAHIRI, 0, 0)

    # 1. Find time of sunrise
    lat = place[1]
    lon = place[2]
    tz = place[3]
    rise = sunrise(jd, place)[1] - (tz/24)

    # Swiss Ephemeris always gives Sayana. So subtract
    # ayanamsa to get Nirayana
    offsets = c(0, 0.25, 0.5, 0.75, 1)
    longitudes = c()
    for (i in 1:length(offsets)) {
        longitudes <- append(longitudes, ((moon_longitude(rise +
            offsets[i]) - swe_get_ayanamsa_ex_ut(rise, SE$FLG_SWIEPH +
            SE$FLG_NONUT)$daya)%%360))
    }
    # 2. Today's nakshatra is when offset = 0 There are 27
    # Nakshatras spanning 360 degrees
    nak = ceiling(longitudes[1] * 27/360)

    # 3. Find end time by 5-point inverse Lagrange
    # interpolation
    y = unwrap_angles(longitudes)
    x = offsets
    approx_end = inverse_lagrange(x, y, nak * 360/27)
    ends = (rise - jd + approx_end) * 24 + tz
    answer = c(as.integer(nak), to_dms(ends))

    # 4. Check for skipped nakshatra
    nak_tmrw = ceiling(longitudes[length(longitudes) - 1] * 27/360)
    if (((nak_tmrw - nak)%%27) > 1) {
        leap_nak = nak + 1
        approx_end = inverse_lagrange(offsets, longitudes, leap_nak *
            360/27)
        ends = (rise - jd + approx_end) * 24 + tz
        answer <- append(answer, c(as.integer(leap_nak), to_dms(ends)))
```

```
    }
    return(answer)
}
```

The **Ayanamsa** which is obtained from the *Swiss Ephemeris* and is set to *Lahiri Ayanamsa*. **Date** as Julian's day number and the **place** as latitude, longitude, and time zone are passed as arguments of the function. The **sunrise** for the given date and place is found using the sunrise function which uses the *Swiss Ephemeris* to get the sunrise as a Julian day number. The *Nirayana longitude* of the **Moon** is obtained by subtracting the *Ayanamsa* from its *Sayana longitude*. The *Nirayana longitude* of moon is multiplied by 27 and then divided by 360 to get Nakshatra as a decimal number. The ending time of this Nakshatra is obtained by checking for the starting time of the next Nakshatra using *Lagrange's inverse interpolation* of Nirayana lunar longitudes at 0.25 intervals of day. Nakshatra and its ending time are added to the output vector. Sometimes two Nakshatras can occur on a given date so, we check for skipped Nakshatra by checking the Nakshatra of the next day and if a Nakshatra is skipped then, the skipped Nakshatra is included in the output with its ending time.

Nakshatra for given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
nakshatra(jd, place)
#> [1] 25 24 24  1
```

```
nakshatra(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] 21  9 56 11
```

Nakshatra's name

The following are **27 Nakshatras** with their corresponding numbers -

```
#> [1] "1  -  Ashwini"
#> [1] "2  -  Bharani"
#> [1] "3  -  Kritika"
#> [1] "4  -  Rohini"
#> [1] "5  -  Mrigashira"
#> [1] "6  -  Ardra"
#> [1] "7  -  Punarvasu"
#> [1] "8  -  Pushya"
#> [1] "9  -  Ashlesha"
#> [1] "10  -  Magha"
#> [1] "11  -  Purvaphalguni"
#> [1] "12  -  Uttaraphalguni"
#> [1] "13  -  Hasta"
#> [1] "14  -  Chitra"
#> [1] "15  -  Swati"
#> [1] "16  -  Vishakha"
#> [1] "17  -  Anuradha"
#> [1] "18  -  Jyeshta"
#> [1] "19  -  Mula"
#> [1] "20  -  Purvashada"
#> [1] "21  -  Uttarashada"
#> [1] "22  -  Shravana"
#> [1] "23  -  Dhanishta"
#> [1] "24  -  Shatabhisha"
#> [1] "25  -  Purvabhadrapada"
#> [1] "26  -  Uttarabhadrapada"
```

```
#> [1] "27  -  Revati"
```

Get name(s) of the Nakshatra for given Julian day number and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
get_nakshatra_name(jd, place)
#> [1] "Purvabhadrapada till 24:24:1"
```

```
get_nakshatra_name(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))
#> [1] "Uttarashada till 9:56:11"
```

Yoga

There are **27 Yogas**, each measure 13°20' of the arc. Yoga is the sum of sidereal (*Nirayana*) longitudes of sun and moon in the multiples of 13° 20'. When the sidereal longitudes of sun and moon are added and they are divided by 13°20', the quotient plus one gives the number which denotes the respective Yoga.

Calculation of Yoga

```
yoga <- function(jd, place) {
    # Yoga as -> 1 = Vishkambha, 2 = Priti, ..., 27 =
    # Vaidhrti
    swe_set_sid_mode(SE$SIDM_LAHIRI, 0, 0)

    # 1. Find time of sunrise
    lat = place[1]
    lon = place[2]
    tz = place[3]
    rise = sunrise(jd, place)[1] - (tz/24)

    # 2. Find the Nirayana longitudes and add them
    lunar_long = (moon_longitude(rise) - swe_get_ayanamsa_ex_ut(rise,
        SE$FLG_SWIEPH + SE$FLG_NONUT)$daya)%%360
    solar_long = (sun_longitude(rise) - swe_get_ayanamsa_ex_ut(rise,
        SE$FLG_SWIEPH + SE$FLG_NONUT)$daya)%%360
    total = (lunar_long + solar_long)%%360

    # There are 27 Yogas spanning 360 degrees
    yog = ceiling(total * 27/360)

    # 3. Find how many longitudes is there left to be swept
    degrees_left = yog * (360/27) - total

    # 4. Compute longitudinal sums at intervals of 0.25
    # days from sunrise
    offsets = c(0.25, 0.5, 0.75, 1)
    lunar_longitude_diff = c()
    solar_longitude_diff = c()
    total_motion = c()

    for (i in 1:length(offsets)) {
        lunar_longitude_diff <- append(lunar_longitude_diff,
            ((moon_longitude(rise + offsets[i]) - moon_longitude(rise))%%360))
```

```
        solar_longitude_diff <- append(solar_longitude_diff,
            ((sun_longitude(rise + offsets[i]) - sun_longitude(rise))%%360))
        total_motion <- append(total_motion, (lunar_longitude_diff[i] +
            solar_longitude_diff[i]))
    }
    # 5. Find end time by 4-point inverse Lagrange
    # interpolation
    y = total_motion
    x = offsets
    # compute fraction of day (after sunrise) needed to
    # traverse 'degrees_left'
    approx_end = inverse_lagrange(x, y, degrees_left)
    ends = (rise + approx_end - jd) * 24 + tz
    answer = c(as.integer(yog), to_dms(ends))

    # 5. Check for skipped yoga
    lunar_long_tmrw = (moon_longitude(rise + 1) - swe_get_ayanamsa_ex_ut(rise +
        1, SE$FLG_SWIEPH + SE$FLG_NONUT)$daya)%%360
    solar_long_tmrw = (sun_longitude(rise + 1) - swe_get_ayanamsa_ex_ut(rise +
        1, SE$FLG_SWIEPH + SE$FLG_NONUT)$daya)%%360
    total_tmrw = (lunar_long_tmrw + solar_long_tmrw)%%360
    tomorrow = ceiling(total_tmrw * 27/360)
    if (((tomorrow - yog)%%27) > 1) {
        # interpolate again with same (x,y)
        leap_yog = yog + 1
        degrees_left = leap_yog * (360/27) - total
        approx_end = inverse_lagrange(x, y, degrees_left)
        ends = (rise + approx_end - jd) * 24 + tz
        answer <- append(answer, c(as.integer(leap_yog), to_dms(ends)))
    }
    return(answer)
}
```

The **Ayanamsa** which is obtained from the *Swiss Ephemeris* and is set to *Lahiri Ayanamsa*. **Date** as Julian's day number and the **place** as latitude, longitude, and time zone are passed as arguments of the function. The *sunrise* for the given date and place is found using the sunrise function which uses the *Swiss Ephemeris* to get the sunrise as a Julian day number. The *Nirayana* longitudes of the sun and moon are obtained by subtracting *Ayanamsa* from their *Sayana longitudes*. The sum of *Nirayana longitudes* of **Moon** and **Sun** is obtained and their mod 360 is performed so that the sum does not exceed 360. The resulting sum is multiplied by 27 and then divided by 360 to get Yoga as a decimal number. The fractional part of the Yoga is used to find out the remaining degrees to the next Yoga which is used to find the time when that Yoga ends. The time taken to complete the remaining degrees is given by *Inverse Lagrange's interpolation* of the Yoga found by using the lunar and solar longitudes at the intervals of 0.25 days from sunrise. The obtained Yoga and its ending time are added to the output vector. Sometimes two Yogas can occur on a given date so, we check for skipped Yoga by checking the Yoga of the next day and if a Yoga is skipped then, the skipped Yoga is included in the output with its ending time.

Yoga for given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
yoga(jd, place)
#> [1]  5 27 26 12
```

```
yoga(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] 26 17 17 57
```

Yoga's name

The following are **27 Yogas** with their corresponding numbers -

```
#> [1] "1  -  Vishkhamba"
#> [1] "2  -  Preeti"
#> [1] "3  -  Ayushmaan"
#> [1] "4  -  Saubhaagya"
#> [1] "5  -  Sobhana"
#> [1] "6  -  Atiganda"
#> [1] "7  -  Sukarman"
#> [1] "8  -  Dhriti"
#> [1] "9  -  Shoola"
#> [1] "10  -  Ganda"
#> [1] "11  -  Vriddhi"
#> [1] "12  -  Dhruva"
#> [1] "13  -  Vyaaghaata"
#> [1] "14  -  Harshana"
#> [1] "15  -  Vajra"
#> [1] "16  -  Siddhi"
#> [1] "17  -  Vyatipaata"
#> [1] "18  -  Variyan"
#> [1] "19  -  Parigha"
#> [1] "20  -  Shiva"
#> [1] "21  -  Siddha"
#> [1] "22  -  Saadhya"
#> [1] "23  -  Subha"
#> [1] "24  -  Sukla"
#> [1] "25  -  Brahma"
#> [1] "26  -  Indra"
#> [1] "27  -  Vaidhriti"
```

Get name(s) of the Yoga for given Julian day number and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
get_yoga_name(jd, place)
#> [1] "Sobhana till 27:26:12"
```

```
get_yoga_name(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] "Indra till 17:17:57"
```

Karana

A **Karana** is half of a tithi or when the moon traverses 6° in longitude relative to the sun. In *30 tithis* of a lunar month, there are 60 *Karanas* or half-tithis.

Calculation of Karana

```
karana <- function(jd, place) {
    tithi_ = tithi(jd, place)
    answer <- c((tithi_[1] * 2) - 1, tithi_[1] * 2)
```

```
    return(answer)
}
```

To find out Karana for a given date and place where, **date** is given as a *Julian day number* and **place** as latitude, longitude and time zone. Date and place are passed to the Tithi function which finds out the Tithi(s). This Tithis are used to output corresponding Karnas.

An excerpt from *Elements of Vedic Astrology by Dr. K. S. Charak*

There are **4 Karanas** that occur **only once** in a lunar month. They are the fixed Karanas and called as:

1.*Shakuni*: assigned to the latter half of the 14th day of *Krishna-paksha*.
2.*Chatushpada*: assigned to the first half of the Amavasya (15th day of *Krishna-paksha*).
3.*Naga*: assigned to the latter half of the *Amavasya*.
4.*Kimstughna*: assigned to the first half of the first day of the *Shukla-paksha*.

The remaining **seven Karanas** recur **eight times** during rest of the lunar month. Their names are:

1. *Bava*
2. *Balava*
3. *Kanlava*
4. *Taitila*
5. *Gara*
6. *Vanija*
7. *Visht*

These Karanas recur in regular order starting from the second half of the first day of *Shukla-paksha* until the first half of the 14th day of the *Krishna-paksha*.

Karana for given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
karana(jd, place)
#> [1] 39 40
```

```
karana(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] 35 36
```

Karana's name

Get name(s) of the Karana for given Julian day number and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
get_karana_name(jd, place)
#> [1] "Kaulava-Taitila"
```

```
get_karana_name(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))
#> [1] "Vanija-Visti"
```

Other functions

Rashi

The zodiac is divided into 12 parts - Aries (*Mesha*), Taurus (*Vrishabha*), Gemini (*Mithuna*), Cancer (*Karka*), Leo (*Simha*), Virgo (*Kanya*), Libra (*Tula*), Scorpio (*Vruschika*), Sagittarius (*Dhanu*), Capricorn (*Makara*),

Aquarius (*Kumbha*), and Pisces (*Meena*).

Rashi (moon sign) represents the position of the moon on the zodiac at a given time.

Calculation of Rashi

```
rashi <- function(jd) {
    swe_set_sid_mode(SE$SIDM_LAHIRI, 0, 0)
    s = moon_longitude(jd)
    lunar_nirayana = (moon_longitude(jd) - swe_get_ayanamsa_ex_ut(jd,
        SE$FLG_SWIEPH + SE$FLG_NONUT)$daya)%%360
    return(ceiling(lunar_nirayana/30))
}
```

The *Ayanamsa* is set to *Lahiri ayanamsa* and the lunar longitude(tropical) is obtained for the given date which is taken as the Julian day number. The sidereal (*Nirayana*) longitude of the moon is calculated by subtracting the Ayanamsa from the obtained tropical longitude then a mod of 360 is performed to ensure that it doesn't exceed 360 and the result is divided by 30 to obtain Rashi as an integer.

Rashi(Moon-sign) for a given Julian day number -

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
rashi(jd)
#> [1] 10
```

Rashi's name

The following are **Rashis** with their corresponding numbers -

```
#> [1] "1  -  Mesha"
#> [1] "2  -  Vrushabha"
#> [1] "3  -  Mithuna"
#> [1] "4  -  Karka"
#> [1] "5  -  Sinha"
#> [1] "6  -  Kanya"
#> [1] "7  -  Tula"
#> [1] "8  -  Vrushchik"
#> [1] "9  -  Dhanu"
#> [1] "10  -  Makara"
#> [1] "11  -  Kumbha"
#> [1] "12  -  Meena"
```

Gives name of the Rashi(Sun-sign) for a given Julian day number

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
get_rashi_name(jd)
#> [1] "Makara"
```

Lagna

Lagna (sun sign) represents the position of the sun on the zodiac at a given time.

Calculation of Lagna

```
lagna <- function(jd) {
    swe_set_sid_mode(SE$SIDM_LAHIRI, 0, 0)
    s = sun_longitude(jd)
```

```
    solar_nirayana = (sun_longitude(jd) - swe_get_ayanamsa_ex_ut(jd,
        SE$FLG_SWIEPH + SE$FLG_NONUT)$daya)%%360
    return(ceiling(solar_nirayana/30))
}
```

The *Ayanamsa* is set to *Lahiri ayanamsa* and the solar longitude(tropical) is obtained for the given date which is taken as the Julian day number. The sidereal (*Nirayana*) longitude of the sun is calculated by subtracting the Ayanamsa from the obtained tropical longitude then a mod of 360 is performed to ensure that it doesn't exceed 360 and the result is divided by 30 to obtain Lagna as an integer.

Lagna(sun-sign) for a given Julian day number -

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
lagna(jd)
#> [1] 3
```

Lagna's name

Gives name of the lagna for a given Julian day number

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
get_lagna_name(jd)
#> [1] "Mithuna"
```

Masa

Masa is the lunar month in the Vedic calendar system. The year as per the Hindu calendar starts from *Chaitra masa* and ends at *Phalguna masa*. Every masa has a fixed number of *Tithis*. A Masa has **2 Pakshas** (*Krishna paksha* and *Shukla paksha*) - **30 Tithis** in total.

Chaitra Masa doesn't begin on the 1st January of the Gregorian calendar. It rather begins on *Ugadi* (*Gudi Padva* or Hindu new year) which starts when *Amavasya Tithi* (New moon) of the *Phalguna masa* ends.

In one year, the number of Masas is not fixed as there can be 12 or 13 Masas in it. When the number of Tithis in a Masa is fixed, to compensate for the remaining days (Like in the Gregorian calendar which has 28-31 days in a month) an intercalary month is added every 32.5 months. This extra masa is called *Adhika Masa*. It is usually referred by adding a suffix of its previous masa like – *Adhika Jyeshtha masa*. It can occur between any two Masas but, it follows a regular interval of 32.5 months.

As there is the lunar year with the extra month (*Adhika masa*), so it there a lunar year with a diminished or reduced month, with only eleven months only is very rare indeed. It occurs once in 140 years or once in 190 years it is called *Kshaya Masa* or Lost Month.

Calculation of Masa

```
masa <- function(jd, place) {
    # Masa as -> 1 = Chaitra, 2 = Vaisakha, ..., 12 =
    # Phalguna
    ti = tithi(jd, place)[1]
    critical = sunrise(jd, place)[1]
    last_new_moon = new_moon(critical, ti, -1)
    next_new_moon = new_moon(critical, ti, +1)
    this_solar_month = lagna(last_new_moon)
    next_solar_month = lagna(next_new_moon)
    is_leap_month = (this_solar_month == next_solar_month)
    maasa = this_solar_month + 1
    if (maasa > 12) {
```

```
        maasa = maasa%%12
    }
    return(c(as.integer(maasa), is_leap_month))
}
```

At first, the *Tithi* and sunrise of the given date and place is obtained then, the previous and next new moon is obtained from the given date and calculated *tithi*. The sun sign(*lagna*) for the previous and next new moon day is obtained. If these two sun-signs are same then the Masa is *Adhika masa*. The masa number is one added to the sun sign of this month. Its mod 12 is returned so that it does not exceed 12.

Masa for a given place and time.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
masa(jd, place)
#> [1] 4 0
```

```
masa(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] 3 0
```

Masa's name

The following are **Masas** with their corresponding numbers -

```
#> [1] "1  -  Chaitra"
#> [1] "2  -  Vaishakha"
#> [1] "3  -  Jyeshtha"
#> [1] "4  -  Ashada"
#> [1] "5  -  Shravana"
#> [1] "6  -  Bhadrapada"
#> [1] "7  -  Ashvija"
#> [1] "8  -  Kartika"
#> [1] "9  -  Margashira"
#> [1] "10  -  Pushya"
#> [1] "11  -  Maagha"
#> [1] "12  -  Phalguna"
```

Get name of the Masa for given Julian day number and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
get_masa_name(jd, place)
#> [1] "Ashada"
```

```
get_masa_name(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13, +5.5))
#> [1] "Jyeshtha"
```

Ritu

Ritu means **season**. There are 6 Ritus which represent the 6 seasons. The year starts from *Vasanta ritu* which is the spring season and ends at *Shishir Ritu* which is the winter season.

The following are the Ritus , seasons they represent and their respective Masas -

1. Vasant – Spring (Chaitra and Vaishakha)
2. Grishma – Summer (Jyestha and Ashadha)
3. Varsha – Monsoon (Shravana and Bhadrapada)
4. Sharad – Autumn (Ashwin and Kartika)

5. Hemant – Prewinter (Margashira and Pushya)
6. Shishir – Winter (Magha and Phalguna)

Calculations of Ritu

```
ritu <- function(masa_num) {
    return(((masa_num - 1)%/%2) + 1)
}
```

Returns the number associated with Ritu from a Masa.

```
masa_num <- masa(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))[1]
ritu(masa_num)
#> [1] 2
```

Ritu's name

The following are **Ritus** with their corresponding numbers -

```
#> [1] "1  -  Vasanta"
#> [1] "2  -  Grishma"
#> [1] "3  -  Varsha"
#> [1] "4  -  Sharad"
#> [1] "5  -  Hemanta"
#> [1] "6  -  Sishira"
```

Returns Ritu's name from a Masa.

```
masa_num <- masa(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))[1]
get_ritu_name(masa_num)
#> [1] "Grishma"
```

Samvatsara

Samvatsara means *Year* in *Sanskrit*. It is a cycle of 60 Samvatsaras in Hindu Panchang which is of 60 years. There are different types of Samvatsaras based on the point which is considered as their starting point. *Saka Samvatsara*, the epoch of which corresponds to Julian year 78 and *Vikram Samvatsara* which is ahead of *Saka Samvatsara* by 135 years are some of the commonly used calendar systems.

Calculations of Samvatsara

```
ahargana <- function(jd) {
    return(jd - 588465.5)
}


elapsed_year <- function(jd, maasa_num) {
    sidereal_year = 365.25636
    ahar = ahargana(jd)
    kali = as.integer((ahar + (4 - maasa_num) * 30)/sidereal_year)
    saka = kali - 3179
    vikrama = saka + 135
    return(c(kali, saka, vikrama))
}
```

```
samvatsara <- function(jd, maasa_num) {
    kali = elapsed_year(jd, maasa_num)[1]
    if (kali >= 4009) {
        kali = (kali - 14)%%60
    }
    samvat = (kali + 27 + as.integer((kali * 211 - 108)/18000))%%60
    return(samvat)
}
```

Returns number associated with the name of the Shaka Samvatsar for a given Julian day number and maasa number.

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number

# Number associated with the masa
masa_num <- masa(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))[1]

samvatsara(jd, masa_num)
#> [1] 36
```

Samvatsara's name

The following are **Samvatsars** with their corresponding numbers -

```
#> [1] "1  -  Prabhava"
#> [1] "2  -  Vibhava"
#> [1] "3  -  Sukla"
#> [1] "4  -  Pramoda"
#> [1] "5  -  Prajapati"
#> [1] "6  -  Angirasa"
#> [1] "7  -  Srimukha"
#> [1] "8  -  Bhava"
#> [1] "9  -  Yuva"
#> [1] "10  -  Dhatri"
#> [1] "11  -  Ishvara"
#> [1] "12  -  Bahudhanya"
#> [1] "13  -  Pramadhi"
#> [1] "14  -  Vikrama"
#> [1] "15  -  Vrushapraja"
#> [1] "16  -  Citrabhanu"
#> [1] "17  -  Subhanu"
#> [1] "18  -  Tarana"
#> [1] "19  -  Parthiva"
#> [1] "20  -  Vyaya"
#> [1] "21  -  Sarvajit"
#> [1] "22  -  Sarvadharin"
#> [1] "23  -  Virodhin"
#> [1] "24  -  Vikriti"
#> [1] "25  -  Khara"
#> [1] "26  -  Nandana"
#> [1] "27  -  Vijaya"
#> [1] "28  -  Jaya"
#> [1] "29  -  Manmatha"
```

```
#> [1] "30  -  Durmukhi"
#> [1] "31  -  Hevilambi"
#> [1] "32  -  Vilambi"
#> [1] "33  -  Vikari"
#> [1] "34  -  Sharvari"
#> [1] "35  -  Plava"
#> [1] "36  -  Shubhakrit"
#> [1] "37  -  Shobhakrit"
#> [1] "38  -  Krodhi"
#> [1] "39  -  Vishvavasu"
#> [1] "40  -  Parabhava"
#> [1] "41  -  Plavanga"
#> [1] "42  -  Kilaka"
#> [1] "43  -  Saumya"
#> [1] "44  -  Sadharana"
#> [1] "45  -  Virodhakruta"
#> [1] "46  -  Paridhavi"
#> [1] "47  -  Pramadi"
#> [1] "48  -  Ananda"
#> [1] "49  -  Rakshasa"
#> [1] "50  -  Nala"
#> [1] "51  -  Pingala"
#> [1] "52  -  Kalayukta"
#> [1] "53  -  Siddharti"
#> [1] "54  -  Raudra"
#> [1] "55  -  Durmati"
#> [1] "56  -  Dundubhi"
#> [1] "57  -  Rudhirodhgari"
#> [1] "58  -  Raktakshi"
#> [1] "59  -  Krodhana"
#> [1] "60  -  Akshaya"
```

Returns the name of the Shaka Samvatsar for a given Julian day number and maasa number.

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number

# Number associated with the masa
masa_num <- masa(gregorian_to_jd(17, 6, 2022), c(15.34, 75.13,
    +5.5))[1]

get_samvatsara_name(jd, masa_num)
#> [1] "Shubhakrit"
```

Intermediate functions

The following functions use the functions of **SwephR - Swiss Ephemeris** to return results.

Gregorian to JD

Convert Gregorian date to Julian day number at 00:00 UTC

```
gregorian_to_jd(17, 6, 2022)  #In dd,mm,yyyy
#> [1] 2459748
```

JD to Gregorian

Convert Julian day number to Gregorian date

```
jd_to_gregorian(2459778)
#> $year
#> [1] 2022
#>
#> $month
#> [1] 7
#>
#> $day
#> [1] 17
#>
#> $hour
#> [1] 12
```

Sun's Longitude

Get Solar longitude for a given Julian day number.

```
sun_longitude(2459778)
#> [1] 114.91
```

```
sun_longitude(gregorian_to_jd(17, 6, 2022))
#> [1] 85.815
```

Moon's lonitude

Get Lunar longitude for a given Julian day number.

```
moon_longitude(2459778)
#> [1] 346.57
```

```
moon_longitude(gregorian_to_jd(17, 6, 2022))
#> [1] 301.41
```

Sunrise

Sunrise for a given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
sunrise(jd, place)
#> [1] 2459779      18       8      45
```

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
sunrise(jd, place)
#> [1] 2459748       6       0      19
```

Sunset

Sunset for a given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
```

```
sunset(jd, place)
#> [1] 2459778       7       2      46
```

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
sunset(jd, place)
#> [1] 2459748      19       0      29
```

Moonrise

Moonrise for a given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
moonrise(jd, place)
#> [1] 10 26 24
```

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
moonrise(jd, place)
#> [1] 22 16 18
```

Moonset

Moonset for a given date and place.

```
jd <- 2459778  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
moonset(jd, place)
#> [1] 22 31 46
```

```
jd <- gregorian_to_jd(17, 6, 2022)  #Julian day number
place <- c(15.34, 75.13, +5.5)  #Latitude, Longitude and timezone of the location
moonset(jd, place)
#> [1]  8 45 55
```