Time representation in biological time series

Philippe Grosjean

2024-02-01

Computer representation of time is a difficult topic because it does not fit in a decimal system. Many people (but not all) use the Gregorian calendar. One year is divided into four seasons, and roughly regular 12 months of 28-31 days each. This supperposes more or less with lunar cycles that last 29.5 days. Finally, a week is more or less the quarter of a month, and there are just a little bit more than 52 weeks per year. None of these divisions are a perfect fraction of a year, and the year itself is not a constant number of days, since leap years have 366 days instead of 365.

At the day level, it is easier, although still not decimal. The day as a unit is only slightly shifted from time to time by leap seconds (see `.leap.seconds`). However, *local* time measurement is also regulated by time zones and changes between summer and winter times.

```
.leap.seconds
#>  [1] "1972-07-01 GMT" "1973-01-01 GMT" "1974-01-01 GMT" "1975-01-01 GMT"
#>  [5] "1976-01-01 GMT" "1977-01-01 GMT" "1978-01-01 GMT" "1979-01-01 GMT"
#>  [9] "1980-01-01 GMT" "1981-07-01 GMT" "1982-07-01 GMT" "1983-07-01 GMT"
#> [13] "1985-07-01 GMT" "1988-01-01 GMT" "1990-01-01 GMT" "1991-01-01 GMT"
#> [17] "1992-07-01 GMT" "1993-07-01 GMT" "1994-07-01 GMT" "1996-01-01 GMT"
#> [21] "1997-07-01 GMT" "1999-01-01 GMT" "2006-01-01 GMT" "2009-01-01 GMT"
#> [25] "2012-07-01 GMT" "2015-07-01 GMT" "2017-01-01 GMT"
```

All these oddities are handled by time objects in R, being basic `POSIXct`/`POSIXlt`, or `Dates` objects in base R, or their equivalent in packages like **lubridate**, **hms** and many more. However, in **pastecs** we are working with time series where the interesting parts are *long-term trends* and *cycles*, from a biological/ecological point of view. This is a little bit different than our calendar time. For instance, week days, *versus* week-ends have rather limited effects and interests on ecological series. The same can be said from holidays.

On the other hand, there are three major interesting cycles: the day (circadian cycle), the years (successive seasons), and the moon cycle that influences tides, night-time light and many biological aspects directly or indirectly.

Time in pastecs time series

In **pastecs** we stick with the base **ts** and **mts** objects. A **ts** object is a numerical vector representing measurement done at regular time interval, together with a **tsp** attribute and a `ts` class. See for instance, `nottem`, the temperature in degrees Fahrenheit recorded at Nottingham Castle over 20 years.

```
data(nottem)
nottem
#>      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
#> 1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
#> 1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
#> 1922 37.5 38.7 39.5 42.1 55.7 57.8 56.8 54.3 54.3 47.1 41.8 41.7
#> 1923 41.8 40.1 42.9 45.8 49.2 52.7 64.2 59.6 54.4 49.2 36.3 37.6
#> 1924 39.3 37.5 38.3 45.5 53.2 57.7 60.8 58.2 56.4 49.8 44.4 43.6
#> 1925 40.0 40.5 40.8 45.1 53.8 59.4 63.5 61.0 53.0 50.0 38.1 36.3
#> 1926 39.2 43.4 43.4 48.9 50.6 56.8 62.5 62.0 57.5 46.7 41.6 39.8
#> 1927 39.4 38.5 45.3 47.1 51.7 55.0 60.4 60.5 54.7 50.3 42.3 35.2
#> 1928 40.8 41.1 42.8 47.3 50.9 56.4 62.2 60.5 55.4 50.2 43.0 37.3
#> 1929 34.8 31.3 41.0 43.9 53.1 56.9 62.5 60.3 59.8 49.2 42.9 41.9
#> 1930 41.6 37.1 41.2 46.9 51.2 60.4 60.1 61.6 57.0 50.9 43.0 38.8
#> 1931 37.1 38.4 38.4 46.5 53.5 58.4 60.6 58.2 53.8 46.6 45.5 40.6
#> 1932 42.4 38.4 40.3 44.6 50.9 57.0 62.1 63.5 56.3 47.3 43.6 41.8
#> 1933 36.2 39.3 44.5 48.7 54.2 60.8 65.5 64.9 60.1 50.2 42.1 35.8
#> 1934 39.4 38.2 40.4 46.9 53.4 59.6 66.5 60.4 59.2 51.2 42.8 45.8
#> 1935 40.0 42.6 43.5 47.1 50.0 60.5 64.6 64.0 56.8 48.6 44.2 36.4
#> 1936 37.3 35.0 44.0 43.9 52.7 58.6 60.0 61.1 58.1 49.6 41.6 41.3
#> 1937 40.8 41.0 38.4 47.4 54.1 58.6 61.4 61.8 56.3 50.9 41.4 37.1
#> 1938 42.1 41.2 47.3 46.6 52.4 59.0 59.6 60.4 57.0 50.7 47.8 39.2
#> 1939 39.4 40.9 42.4 47.8 52.4 58.0 60.7 61.8 58.2 46.7 46.6 37.8
attributes(nottem)
#> $tsp
#> [1] 1920.000 1939.917   12.000
#>
#> $class
#> [1] "ts"
```

In a `mts` object, the numeric vector is replaced by a matrix that represents *multiple* measurements done at the same regular time intervals.

The **tsp** attributes indicates time with only three values no mather the length of the series: the **start time**, exprimed with decimal values in *time units*, the **end time**, and the **frequency** (that is, the number of observations per unit of time). So, two questions: (1) what time units? (2) How complex and inheritently non-decimal time measurements can be squashed into decimal values?

Time units

As strange as it may be, time units is *not* indicated nor imposed in **ts** objects. You are free to chose the one you like: second, minute, hour, day, week, month, year, century, etc. But as soon as you chose the unit, you must reexpress the time as a decimal value of this unit.

You are free to choose your time units, but that does not mean all time units are equals. There are better ones… and it depends on what you want to study in your time series. **The best time unit is indeed the one that matches the main cycle you want to study**. Hence, the two key time units are the **day** (for circadian cycles) and the **year** (for seasonal effects).

Days

If you are working with time series that focus on circadian effects, you should use the day as time unit. Then, time is reexpressed as a decimal fraction of a day. For instance, 0.5 is midday, 0.25 is 6 AM, etc. Since time is stored as numeric internally in R, it is *already* converted into decimal time. There is nothing to do, except to understand the concept of "a decimal fraction of a day".

You don't have to worry too much about leap seconds (although, it is useful to document you a little bit about it), but you should be very attentive to time zones, UTC time and all these concepts in order to properly convert you time into decimal fractions of a day. Otherwise, there is not much difficulties here.

Reasonable choices for the frequency of observations are 1 (every day), 2 (every twelve hours), 4 (every 6h), 12, 24, 48 (every 1/2h), 144 (every 10min), 288 (every 5min), or 1440 (every min). Of course, you can use any frequency you like.

> Take care that the **ts** object with `frequency = 12` or `frequency = 4` will automatically assume that the time units is *year* and will print the data as months, or quarters respectively (see the `nottem` example). In the present case, it will be thus wrongly printed!

Years

The **year** unit is to be used when you work with seasonal effects, and interannual changes. This unit is much more problematic and in **pastecs** it is simplified into round sub-units, considering seasons (four subunits), or months (twelve subunits) also roughly equivalent to the other major cycle impacting biology: the moon cycle.

To avoid accumulated lags in long time series of tens of years, one year is considered to be 365.25 days. The function `daystoyears()` transforms your time, expressed in days into a year time unit that respects the convention of each year being exactly 365.25 days.

```
library(pastecs)
# A vector with a "days" time-scale (25 values every 30 days)
my_days <- (1:25) * 30
# Convert it to a "years" time-scale, using 23/05/2001 (d/m/Y) as first value
daystoyears(my_days, datemin = "23/05/2001", dateformat = "d/m/Y")
#>  [1] 2001.389 2001.472 2001.554 2001.636 2001.718 2001.800 2001.882 2001.964
#>  [9] 2002.047 2002.129 2002.211 2002.293 2002.375 2002.457 2002.539 2002.621
#> [17] 2002.704 2002.786 2002.868 2002.950 2003.032 2003.114 2003.196 2003.279
#> [25] 2003.361
```

As we can see here, time is now expressed as a decimal fraction of a year, or 365.25 days. The "natural" values you could choose for frequency are 4 (quarters or seasons), 12 (months), 24 (half-months), 48, 52 ("weeks"), 364 ("days") or 384. But no mather what you choose, it won't fit into **actual** calendar sub-units. For instance, for `frequency = 12`, you have something similar to months but perfectly equally spaced. No mather your month is 28, 29, 30 or 31 days long, in the decimal system used in `ts`objects in **pastecs**, it will always be 1/12 of a "year" (as 365.25 days). It means one "month" is indeed:

```
365.25/12
#> [1] 30.4375
```

That is: 30 days, plus a fraction of a day that happens to be 0.4375! How can we manage that? Taken exactly as defined, it would make a problem because the **hour of the day** will shift from month to month, and year to year (each month does not start at the same hour).

> To avoid the problem of the shift in hour, you must separate the circadian effect and the seasonal effect. In order to *eliminate* the circadian effect out of your data, you can consider measurements always taken at the same hour as a starting point. Another solution is to aggregate your measurements by day. Depending on your variable, the aggregation can be done as mean or median daily value, or min, max, or the sum for frequency of occurence of an event.

Hence, before using `daystoyears()` you must make sure you have *no* daily effect anymore in your data. Now, measurements are often planned on a weekly basis, and there are:

```
365.25/7
#> [1] 52.17857
```

… approximatey 52 weeks (plus 0.179) in a "year". So you have to choose for frequency: either `frequency = 48` as "four measurements per month", but then, time interval is:

```
365.25/48
#> [1] 7.609375
```

seven days **plus 14h and 24min**, or `frequency = 52`, and you got much closer to your "real week", with a difference of *only* roughly 35min. However, the choice of 52 is less optimal in term of months, because 52 does not divides exactly by twelve, while 48 does.

The same dilemna exists for "days". Either you can choose `frequency = 364`, which matches the actual number of days (minus 1.25) and is a multiple of seven (exactly 52 "weeks" in 364 "days"), or you must consider `frequency = 360` to get something as closer as possible to a "day", while remaining a multiple of twelve. And if you want a multiple of 4, 12, 24, and 48 simultaneously, you must consider `frequency = 384`.

So, what?

- In **pastecs**, time is simplified to a decimal fraction of a time unit.
- Best time unit should match the main cycle you are interested in your time series: **days** for circadian cycle, **years** for seasonal cycle.
- With the **days**, you must just understand the notion of "fraction of a day", and take care of the **time zone** in your conversion.
- With the **years** unit, you must understand that everything is not only transformed into decimal fraction of a year, but also that all units and subunits are **exact divisions**, and thus, they are **not** perfect matches of real sub-units in the Gregorian calendar like months, weeks, or days. It makes sense in the context of methods and tools used on time series to have exact subunits, plus, usually the living organisms do not really care if we are sunday or monday!
- Conversion from time expressed in days must be done with `daystoyears()` to take this into account, but only after making sure that the daily effect is eliminated (only use measurments made at the same hour, or aggregate data by days before transformation), because there will be a *shift in the hours* during the calculation. You can regularize you time series using `regul(units = "daystoyears")` to let **pastecs** manage all the gory details.
- It is convenient to adjust the frequency of observations into a round fraction of the year, with most useful values being 4 (quarters or seasons), 12 (months), 24, 48,… down to 384, which is the closest match to a "day", while remaining divisible by 4, 12, 24 and 48.
- Another division of the year is based on the week (sampling campaigns are often week-based). The closest integer division of the year is then `frequency = 52` for "weeks", or `frequency = 364` for "days". But then, it is **not** perfectly divisible by the major year subunits like "months". The closest match for the "days" to be divisible by twelve is `frequency = 360`.
- You should consider if you really *need* a frequency larger that 48 (or 52) to study interannual variations and seasonal cycles. It is often much better to aggregate data to `frequency = 48`, or `frequency = 52` for "weeks", instead of using one day or less as time interval because the extra data you got that way do not provide useful information at the year-unit level for, say, multi-decanal time series.

> Flattened time in **pastecs** slightly distords the complex reality, but it has no noticeable impact on subsequent analyses of most biological/ecological time series. On the other hand, it greatly simplifies and speeds up calculations in a context where the lag of a time series (a shift in time by a constant value) is very often used in the computations.