DEMOGRAPHIC RESEARCH
A peer-reviewed, open-access journal of population sciences

DEMOGRAPHIC RESEARCH

VOLUME 28, ARTICLE 23, PAGES 649-680
PUBLISHED 27 MARCH 2013
http://www.demographic-research.org/Volumes/Vol28/23/
DOI: 10.4054/DemRes.2013.28.23

Research Material

Chronological objects in demographic research

Frans Willekens

© 2013 Frans Willekens.

This open-access work is published under the terms of the Creative Commons Attribution NonCommercial License 2.0 Germany, which permits use, reproduction & distribution in any medium for non-commercial purposes, provided the original author(s) and source are given credit.
See http:// creativecommons.org/licenses/by-nc/2.0/de/

Table of Contents

1	Introduction	650
2	Chronological objects: overview	653
3	Objects of class “Date” or class POSIX	658
4	Date as elapsed time since reference point in time	662
5	Date as elapsed time since reference event	665
6	Illustrative applications	667
7	Conclusion	671
8	Acknowledgements	672
	References	673
	Annex	678

Demographic Research: Volume 28 Article 23
Research Material

Chronological objects in demographic research

Frans Willekens[1]

Abstract

BACKGROUND
Calendar time, age and duration are chronological objects. They represent an instant or a time period. Age and duration are usually expressed in units with varying lengths. The number of days in a month or a year depends on the position on the calendar. The units are also not homogeneous and the structure influences measurement. One solution, common in demography, is to use units that are large enough for the results not to be seriously affected by differences in length and structure. Another approach is to take the idiosyncrasy of calendars into account and to work directly with calendar dates. The technology that enables logical and arithmetic operations on dates is available.

OBJECTIVE
To illustrate logical and arithmetic operations on dates and conversions between time measurements.

METHOD
Software packages include utilities to process dates. I use existing and a few new utilities in R to illustrate operations on dates and conversions between calendar dates and elapsed time since a reference moment or a reference event. Three demographic applications are presented. The first is the impact of preferences for dates and days on demographic indicators. The second is event history analysis with time-varying covariates. The third is microsimulation of life histories in continuous time.

CONCLUSION
The technology exists to perform operations directly on dates, enabling more precise calculations of duration and elapsed time in demographic analysis. It eliminates the need for (a) approximations and (b) transformations of dates, such as Century Month Code, that are convenient for computing durations but are a barrier to interpretation. Operations on dates, such as the computation of age, should consider time units of varying length.

[1] Netherlands Interdisciplinary Demographic Institute. E-mail: willekens@nidi.nl.

1. Introduction

A person born on July 12th 1977 marries on July 12th 2007. What is the age at marriage? This is a simple question for humans but a complicated one for computers. Software packages do not always produce correct results. The common method is to divide the age in days by the average length of a year. The age at marriage is 10957 days, which translates in age 30 in completed years (exact age of 30.02 years) if the length of a year is taken as 365 days and age 29 (exact age of 29.9986 years) if the more precise approximation of 365.25 days is used. A person born one year later on July 12th 1978 marrying on July 12th 2008 marries at an age of 10958 days, which results in 30 years in both cases. This approach to calculating age, although approximate, is recommended by major texts on survival analysis (e.g. Kalbfleisch and Prentice 2002; Therneau and Grambsch 2000, p. 71). SPSS (IBM 2011, p. 153), STATA (Stata 2009, p. 338) and SAS (SAS Institute, 2008, p. 4663) determine age in days and calculate the age in years by dividing the number of days by 365 (SAS) and 365.25 (SPSS, Stata). The need for an approximation is a consequence of peculiarities of our calendar: years (and months) vary in length. The relevance of this problem from a demographic perspective is twofold. First, demography has historically been particularly concerned about age allocation and demands that, technology permitting, age be computed precisely. Second, the relation between period (current date), cohort (date of birth) and age, which is widely used in demography, is not valid unless age is measured precisely and not approximately.

The calendar has other consequences. A month has a varying number of days of the week. Most months have 4 Saturdays but some have 5 Saturdays. The number of weekends varies too. If death, childbirth, marriage or another demographic event is concentrated on a particular day of the week, then the event is more likely in months with five occurrences of that day than in months with four occurrences. Examples of concentration of events on particular days are widespread. Saturday is a preferred day for weddings (e.g. Haskey 1996). Births are less likely on weekends (e.g. Goodman et al. 2005; Lerchl and Reinhard 2008). Delivering a child outside the normal working week is associated with increased risk of perinatal and neonatal death (e.g. Hong et al. 2006; Pasupathy et al. 2010). The neonatal mortality rate is higher on weekends compared to weekdays (e.g. Salihu et al. 2012). Sudden Infant Death Syndrome (SIDS) is more common on weekends (e.g. Mooney et al. 2004). Patients admitted to hospital on a Saturday or Sunday are more likely to die than patients admitted mid-week (e.g. Freemantle et al. 2012; Mohammed et al. 2012). In some countries, intoxication-related deaths peak during weekends and around festival days when alcohol is widely consumed in excess (e.g. Mäkelä et al. 2005). In the USA, mortality from natural causes spikes around Christmas and New Year (e.g. Phillips et al. 2010). These are some

examples from the literature that demonstrate the role of the calendar in the timing of demographic events.

Calendar time, age, period, interval and duration are key concepts in demography. They are chronological objects. Measurement of these objects and arithmetic and logical operations are common tasks of a demographer. Age and durations are often expressed in years and months, and sometimes in days. Years and months are time units that vary in length. The common approach is to replace the unit of *varying length* with a unit of *fixed length*. Common assumptions of years of fixed length (365 days disregarding leap years, or 365.25 days or 365.242199 days considering leap years) and months of fixed length of 30.437 days are usually good enough approximations. An accurate measurement of age on a particular date from the date of birth is not possible without a calendar. To obtain estimates of demographic rates by month, some authors control for different lengths of months, e.g. Doblhammer and Vaupel (2001) in a study of the effect of month of birth on the lifespan and Sobotka et al. (2005) in a study of fertility levels. The peculiarities of our calendar are particularly significant in simulations of life histories in continuous time.

Units of time used in demography not only vary in length, they are also not homogenous. Their structure impacts measurement. Demographic events are seldom uniformly distributed during the week, month or year. The effect of disregarding preferences for particular years, months or days of the week may distort demographic findings. Many Chinese couples chose to marry or have a child during the year of the Dragon. In the West, Saturday is a preferred day for weddings; it is Friday in other parts of the world. If such preferences exist, the number of Saturdays (and Fridays) in a month may have an effect on the monthly marriage rate. Although the effect is likely to be small, it may be significant. For instance, if 10 percent of the population at risk marries within a year and 80 percent of marriages take place on a Saturday, then the monthly marriage rate is 0.0101 if the month has 5 Saturdays and 0.0087 if the month has 4 Saturdays. Calot (1981), Calot and Sardon (2004), Wilson and Smallwood (2007) and others draw attention to that problem and control monthly marriage rates for day-of-the-week effect. Sobotka et al. (2005) correct monthly birth data for seasonal and calendar factors. Day-of-the-week effects reveal important heterogeneity in demographic processes that remains hidden in demographic studies that use large time units. In the West, it is not uncommon for between 60 and 80 percent of marriages to take place on a Saturday. In the USA between 2005 and 2010, 56% fewer children were born on a Sunday than on a Tuesday or another normal working day[2]. Some effects raise public concern. Using data on 14.2 million patients who were admitted to NHS hospitals in England between April 2009 and March 2010, Freemantle et al. (2012)

[2] Calculated from data in National Vital Statistics Reports 2005-2010,US National Center for Health Statistics, Hyattsville, MD.

show that those admitted on a Sunday have a 16% higher risk of dying within a month than those admitted on a Wednesday. Those who become inpatients on a Saturday are 11% more likely not to survive. The results received wide media attention in the UK and increased the awareness of day-of-the-week effects (see e.g. NHS-London 2011). The measurement of effects of day-of-the-week, public holidays or other peculiarities of our calendar, require operations on calendar dates. The use of chronological objects in data analysis is not new (see e.g. James and Pregibon 1993). What is new is that new technology enables accurate computations on calendar dates. Using this technology to compute ages and durations more precisely is the subject of this paper.

Dates may be represented as calendar dates (year, month and day) or as number of days or months since a reference point in time or a reference event. Several major surveys such as the Demographic and Health Survey (DHS), the US Health and Retirement Survey (HRS), and the Framingham Heart Study (FHS) do not use the Gregorian calendar, but represent dates as times elapsed since a reference point. If the elapsed time is measured in days the date is known as Julian date. The DHS and the US National Survey on Family Growth present dates in Century Month Code (CMC), which counts the number of months since 1st January 1900. The HRS and FHS use SAS dates, which count the number of days since 1st January 1960. Scientists often use *decimal date*, in particular decimal year, which represents a date as the year and fraction of a year. For instance, exposure time in person years and the life expectancy are generally expressed in decimal years. The different representations of dates call for conversion methods. Several conversion methods are addressed in this paper.

To compute durations, R is used. R is an open-source software language for statistical modeling and analysis[3]. It is a collection of functions that perform specific tasks. Basic functions are included in Base R maintained by a Core Team of experts. Additional functions in packages are contributed by researchers around the world and are included in the Comprehensive R Archive Network (CRAN). The date functions presented in this paper are from Base R and contributed packages, in particular *lubridate* (Grolemund and Wickham 2011, 2012) and *Biograph* (Willekens 2012).

The paper consists of seven sections. Section 2 is an introduction to chronological objects. The different types or classes of chronological objects are presented. Section 3 covers one particular class of object more extensively; the object of class "Date", which represents the date in the Gregorian calendar. Dates as elapsed times (days, months or years) since a *reference date* are discussed in Section 4, where Julian dates and CMCs are covered. Dates as elapsed times since a *reference event* are covered in Section 5. Age belongs to this category. Section 6 includes three illustrative applications of date arithmetic in demographic analysis. The first application requires the computation of the number of Saturdays and Sundays in a month. The second is episode splitting, a

[3] www.r-project.org.

common problem in event history analysis with time-varying covariates. The third is the microsimulation of life histories. Section 7 concludes the paper.

2. Chronological objects: overview

A calendar is a system of keeping time. The calendar in use in most of the Western world is the Gregorian calendar, introduced in 1582 by Pope Gregory XIII. It succeeded the Julian calendar, introduced in 46 BC by Julian Caesar. In both calendars a regular year has 365 days divided into 12 months. A leap day is added to February every fourth year. The Julian year is, on average, 365.25 days long. The calendar gave every fourth year 366 days. Reform was required because the Julian calendar drifted behind the solar calendar, based on the Earth's orbital period. At the time of reform the difference was about a week. The Gregorian calendar provides that every fourth year has 366 days, except for years divisible by 100 but not 400. In leap years, it has an extra day in February. The year 2000 is a leap year, but 2100 is not. The new calendar was introduced on October 15, 1582. In the Chinese, the Hindu, and the Hebrew calendars, which are lunisolar calendars, a leap year has an extra month.

On computers and often in science, a date is represented as the time elapsed since a reference point in time to facilitate arithmetic operations on dates. Character variables, generally used to represent dates, are difficult to work with. Time elapsed is a numeric variable and that class of variables enables arithmetic operations. Julian dates and Century Month Codes are examples of dates expressed as time elapsed. The Julian date (JD) system of time measurement presents the time in days and fractions of a day since January 1, 4713 BC Greenwich noon in the Julian calendar. It is used in astronomy. Today, the term *Julian date* or Julian Day Number is used to denote the interval of time in days since a reference date. The reference date used as origin of time measurement is known as (reference) *epoch.* Unix time, or POSIX ("Portable Operating System Interface") time is part of the POSIX standard. POSIX is an open operating system interface specified by the Institute of Electrical and Electronics Engineers (IEEE) to assure code portability between operating systems. POSIX time defines time as the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of Thursday, January 1, 1970. That reference date and time is known as Unix epoch and the POSIX standard is a formula for calculating seconds since the epoch. Most software packages use the Julian date system to store dates and time internally on the computer, but the epoch differs. R uses Unix time. SAS, S-plus and STATA use a different reference: January 1, 1960. An event that occurs on January 30, 1960 occurs at 29 days (1+29). An event that occurs on December 3, 1959 occurs at day –29 (December 31 is day –1 and December 3 is day –29). In Excel, dates are stored as days since January 1,

1900. In SPSS dates and times are stored as numbers of seconds from midnight, October 14, 1582 (the beginning of the Gregorian calendar). For example, in R, March 7, 2002 is represented as 11753. In SAS or Stata, it is 15406, while in SPSS midnight of March 7, 2002 is 13 234 835 972. Dates that occur before the reference date are negative numbers. Julian dates may be used to convert SPSS dates into R, SAS or STATA dates. The number of days since October 14, 1582 is the number of seconds divided by 86400, which is the number of seconds in a day.

Some observational studies also use the Julian date system and present a date as time elapsed since a reference date. For instance, the Framingham Heart Study (FHS), which is a longitudinal study that started in 1948-50 and is widely used in epidemiology, and the US Health and Retirement Study (HRS) use SAS dates (days since January 1, 1960). In the HRS, if the day of an event (e.g. birth) or the day of censoring is not known but the month is recorded, the 15th is the imputed day. If the date of birth is not known but the year is given, 1st July is imputed (see St.Clair et al. 2011, p. 107). Many surveys do not record the day of an event but the month and report the date as the number of months elapsed since a reference date. The Century Month Code is such a form of date reporting. The date is reported as the number of months since 1st January 1900, with January 1900 defined as month 1. The CMC coding scheme is designed to facilitate the computation of elapsed time. Dates in CMC are not directly interpretable. As a consequence, the identification of anomalies and outliers is not straightforward.

Packages in the Comprehensive R Library Network (CRAN) that are particularly useful in demographic research generally require dates to be numeric, i.e. years, months or days since an origin or epoch. The packages include *survival* for survival analysis, *mstate* and *msm* for multistate survival analysis and *Epi* for epidemiological analysis, including drawing Lexis diagrams and age-period-cohort analysis. Date variables that are not numeric "will cause some of the utilities to crash" (Epi manual). The function `cal.yr` of the *Epi* package converts a Date object into decimal year. It first computes the Unix date (Julian date with origin 1st January 1970) and obtains the number of years by dividing the number of days by 365.25. The author, Carstensen (2012) acknowledges that inaccuracies may arise and gives as an illustration 1 Jan 2000. The `cal.yr` function converts that date into 1999.999. Such an inaccuracy can be avoided by using computations that account for varying lengths of months and years.

In R calendar dates are objects of a particular class. The concept of class is central to object-oriented programming. Class is an attribute of an object. Objects in a class are recognized because they share characteristics. Classes direct and streamline operations on objects. They invoke the appropriate method when operations are performed on objects and functions are called to invoke specific operations. If a generic function is applied to an object, the class of the object determines which method is used. For

instance, the generic `plot` function does different things depending on the class of the object to be plotted. When a generic function is called, an internal dispatching method finds the class of the object and invokes the appropriate method. Classes of objects commonly encountered in R include "numeric", "character", "logical", "list", "function" and "Date". In addition, there are many classes defined in packages and user-defined classes. If a date is given as a character string, the object is of class "character". To differentiate it from other character objects, the subclass "date" is given. Dates are usually entered on the computer as character strings. To perform operations on dates, objects of class "character" need to be converted into objects of class "date". R has several functions that convert character strings into date objects. The `as.Date` and `POSIXlt/POSIXct` functions are the most common (see below). The POSIX class is particularly useful when timezone manipulation is important. To facilitate arithmetic and logical operations on dates, the computer converts dates into numeric objects (days or seconds since a reference date).

Base R has a number of functions to process dates. For instance, the `as.Date` function converts a date from a character string into a Date object. `ISOdate` converts three numeric values (year, month and day) into a Date object. The function `weekdays` identifies which day of the week a given calendar date is. In addition, there are several contributed packages that may be used to process dates. The package *chron*, developed by James in S and ported to R by Hornik, handles chronological objects (James and Hornik 2011). It converts between Julian and calendar dates, finds weekdays and holidays, extracts the year, month and day of a date, etc. The package *date* developed by Therneau in S and ported to R by Lumley, Halvorsen and Hornik, has functions for handling dates. It converts calendar dates into Julian dates and vice versa (Therneau et al. 2012). The default reference date is 1st January 1960. The Julian date for June 23, 1965 is 2000 in the *date* package and -1653 in Base R. Several functions of the *date* package were included in earlier versions of the *survival* package. The *survival* package, developed by Therneau and ported to R by Lumley, has a function (`ratetableDate`) that converts an object of class "Date" into number of days since 1st January 1960 (Therneau and Lumley 2012). The function is used in the function `pyears` of the *survival* package, which computes the duration of follow-up by a cohort of subjects. Since the time unit is not variable, the authors recommend using day as the unit of time and converting days into years by dividing the number of days by 365.25. The *ConvCalendar* package, developed by Gray and Lumley (2010), converts between the Date class and d/m/y for several calendars, including Persian, Islamic, and Hebrew. Wuertz et al. (2012) developed the package *timeDate* for financial engineering and computational finance, with special attention to special days, holidays and Daylight Saving Time (see also Chalabi et al. 2011). Grolemund and Wickham (2011, 2012) published *lubridate*.

The package *lubridate* deals with a critical issue in operations on dates; namely, the variable length of the time units month and year. Grolemund and Wickham (2011) distinguish between units of time of constant length (duration is always the same) and *relative* units of time; their length varies and is relative to when they occur. A year may have 365 or 366 days depending on when it occurs relative to other years. Only seconds have a consistent length. Grolemund and Wickham introduce four time-related object classes based on the Java language Joda-Time project (Colebourne and O'Neill 2010). Joda-Time introduces a conceptual model of the different ways Java handles timespans and changes in date and time. The four time-related objects are relevant for demographic analysis. They are: instant, interval, duration and period. An instant is a specific moment in time, such as midnight, January 1, 2013. An instant is defined as the number of seconds from midnight, January 1, 1970 (Joda-Time uses milliseconds). *Lubridate* does not create a new class of instant objects. Instead, it recognizes any date-time object that refers to a moment of time as an instant. It accepts objects of class POSIXct, POSIXlt and "Date" to define instants. For instance, the date object created using `z=as.Date("2010-01-30")` is an instant object. To test it, use `as.instant(z)`. Intervals, durations and periods are ways of recording timespans. An interval is the timespan between two instants. An interval has a length and a position in the calendar. The length of an interval can be determined unambiguously because we know when it occurs. An object of class interval is created by specifying two instants, e.g.

```
library (lubridate)
span <- new_interval(as.Date("2009-01-30"),
                     as.Date("2012-02-25")).
```

Intervals are implemented in *lubridate* (and Joda-Time) as *half-open*, which is to say that the start instant is inclusive but the end instant is exclusive. For age to be measured correctly, age must be an object of class interval. Only in that case does the age account for the varying lengths of months and years.

A period represents the length of an interval. It tracks changes in clock time between two instants. Periods are measured in common time units: years, months, days, hours, minutes, and seconds. With the exception of seconds, none of these units have a fixed length. Leap years and Daylight Saving Time can expand or contract a unit of time depending on when it occurs. For this reason, periods do not have a fixed length unless they are paired with a start date (see below). The following code changes an object of class interval to an object of class period:

```
p=as.period(span,unit="year").
```

The length of the period is measured in year, month, day, hours, minutes and seconds. It is `"3y 0m 26d 0H 0M 0S"`. The object `p` of class period has 6 slots. A slot extracts an object from a class. The year is extracted by `p@year`, the month by `p@month` and the day by `p@day`. The *lubridate* functions `seconds`, `days`, `weeks`, `months` and `years` are shortcuts to create period objects. These units are not of fixed length because units expand or contract in length to accommodate conventions such as leap years, leap seconds, and Daylight Saving Time. Consider a person born on July 12, 1977 who marries at exactly age 30. The date of marriage is the sum of the age at birth, which is an instant (start date), and age, specified as an object of class period:

```
as.Date("1977-07-12") + years(30)
```

which is July 12, 2007. The period object `years(30)` is "30y 0m 0d 0H 0M 0S". The number of days in the 30 years is not fixed but depends on the date of birth.

Duration measures the exact number of seconds in an interval. The duration of the period object 30 years is

```
dur <- as.duration (years (30))
```

The object `dur` is a numeric object and its value is 946080000 seconds. The command `duration (1,"days")` gives the number of seconds in a normal day (86400) and `duration (1,"years")`gives the number of seconds in a year that is not a leap year. The command `duration (span)` gives 96854400s (seconds). It measures the exact passage of time in seconds. The number of seconds does not always align with measurements made in larger units of time such as hours, months and years. This is because the exact length of larger time units are affected by conventions such as leap years and Daylight Saving Time. Seconds are converted into minutes, hours and days using the most common lengths in seconds: Minutes = 60 seconds, hours = 3600 seconds, days = 86400 seconds. The duration of the interval `span` in days is

`duration(span)/duration(1,"days")`, which yields 1121 days. Units larger than days are not used in *lubridate* due to their variability. Values are given with the warning that they are only estimates. The value of `dur` in seconds is exact. The value in years is approximate (~29.98 years).

Base R measures duration with the "difftime" class. The code

```
span <- as.Date("2012-02-25") - as.Date("2009-01-30"))
```

produces an object of class "difftime" representing the number of days between two instants (1121 days).

To determine the exact age of a person on a given date in years (decimal years), the date of birth must be provided. In other words, an object of class interval must be given. In addition, the Gregorian calendar between the two instants must be considered. The length of the timespan between date of birth and the given date is not sufficient to determine the exact age and, as the example at the beginning of this paper shows, to correctly determine the age in completed years. The length of the timespan *in days*, in combination with the date of birth, is sufficient to determine the exact age. Consider a person born on July 12, 1977 who marries on July 12, 2007. The correct age at marriage is

```
f <- new_interval (as.Date ("1977-07-12"),
                   as.Date("2007-07-12"))
z <- as.period (f).
```

The result is "30y 0m 0d 0H 0M 0S". The age in days is `age.d <- duration (f)/duration (1,"days")`, which is 10957 days. The number of years, assuming years of 365.25 days, is `age.d/365.25` or 29.99863 years.

3. Objects of class "Date" or class POSIX

The `as.Date` function of Base R converts a character vector of calendar dates into an object of class "Date". For instance, `z <- as.Date ("2010-08-20")` and `as.Date(paste(2010, 8, 20, sep="-"))` produce an object of class Date. The function `as.numeric(z)` gives the Julian date. For example, `as.numeric(as.Date ("2010-08-20"))` gives 14841, which is the number of days since January 1, 1970. To convert the numeric object into a Date object, use `as.Date(14841,origin="1970-01-01")`. The epoch or origin must be provided.

R stores dates as Unix dates. It communicates the dates with the user by giving the year, month and day. The default format is a four digit year, followed by a month, then a day, separated by either dashes or slashes (e.g. *YYYY-MM-DD* where YYYY is the year, MM is the month and DD is the day). The default format follows the rules of the ISO 8601 international standard which expresses a day as "2001-02-03". Dates do not need to be in the standard format. The as.Date function allows a variety of input formats. For example, `as.Date('20/08/2010',format='%d/%m/%Y')` reads the date as a character string with day, month and four-digit year. The outcome is the date in the standard format: 2010-08-20. The same result is obtained using a different format: `as.Date("August 20 2010",format="%B%d%Y")`. The following symbols can be used with the date format:

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a	abbreviated weekday	Mon
%A	unabbreviated weekday	Monday
%m	month (00-12)	00-12
%b	abbreviated month	Jan
%B	unabbreviated month	January
%y	2-digit year	07
%Y	4-digit year	2007

The class “Date” of Base R is one of several classes of chronological objects. Other classes are "date" (from package *date*), "chron" and "dates" (from package *chron*) and the “timeDate” object of the *timeDate* package. Another important class is the POSIX class mentioned above. The POSIX class is particularly useful when time zones are considered because it allows for different time zones. The `as.Date` function converts a POSIX object into a Date object. The default input format for POSIX dates is the same as Date dates: the year followed by the month and day, separated by slashes or dashes.

Several calculations can be performed on objects of classes “Date” and POSIX. The `ISOdate` function converts a year, month and day into a date object of class POSIXct and class POSIXlt. The POSIXct class represents and stores the date/time values as the number of seconds since January 1, 1970. The POSIXlt class stores them as a list with elements for second, minute, hour, day, month, and year. The year, month and day may be retrieved from this representation. Consider midnight of August 20, 2012. The number of seconds since midnight January 1, 1970 is

```
d <- ISOdate (2012,8,20,0,0,0)
s <- as.numeric(d) = 1345420800
```

where `d` is an object of class POSIXct. If the exact reference time is not specified, the default is noon and not midnight. This default is often confusing in the manipulation of dates in demographic research. The year, month and day can be extracted from the POSIXlt object `f <- as.POSIXlt(d)`. The year is `f$year+1900`; the month is `f$mon+1` (`f$mon` is the number of completed months) and the day is `f$mday`. To extract the components of the date from `s`, use

```
g <- as.POSIXlt(s,origin="1970-01-01").
```

Note that the epoch must be provided. The default time zone is Universal Time, Coordinated (UTC). It is a successor of the Greenwich Mean Time (GMT). GMT differs from London time (British Standard Time BST) because GMT is not affected by Daylight Saving Time. The London time for midnight August 20, 2012 UTC time is

```
as.POSIXlt(s,origin="1970-01-01",tz="Europe/London")
```

which is "2012-08-20 01:00:00 BST".

The function `strptime` also converts a character representation of a date into a POSIXlt object:

```
d <- strptime("20-01-2012",format="%d-%m-%Y").
```

The format of the date needs to be provided. The output format is the ISO format.
Subtraction of two dates gives the number of days between the dates. For instance, the time difference between July 13, 1997 and August 20, 2010 is:

```
b1=ISOdate(1997,7,13,0,0,0)
b2=ISOdate(2010,8,20,0,0,0)
b2-b1
```

which is a time difference of 4786 days. The date is measured at midnight (0 hours, 0 minutes and 0 seconds).

Instead of days, weeks may also be used. In the example above, `difftime(b2,b1,units='weeks')` the time difference is 683.7143 weeks. Units can be seconds, minutes, hours, days or weeks. The input consists of dates of class “Date” or POSIX.
The `format` function is used for formatting dates for output. For example,

```
d<- as.Date("1Mar1946",format="%d%b%Y")
format (d,"%d-%B-%Y")
```

gives us "01-March-1946". To select the year, use `format (date,"%Y")`. An example of this would be :

```
d<- as.Date("1Mar1946",format="%d%b%Y")
format (d,"%Y")
```

which results in the character variable "1946". To obtain the month, use `format (d,"%B"),` to get the day of the month, use `format (d,"%d")` and to get the day of the week, use `format (d,"%A")`. An alternative to the `format` function is the `strptime` function of Base R. It converts a date as a character string into an object of class POSIXlt and gives the date in an ISO date format. For example

```
strptime("25Oct1970",format="%d%b%Y")
```

accepts a date in the format given and returns the date in ISO format. The function strftime accepts a date in the ISO format and returns the date in a desired format. The ISO format is required. Other formats result in wrong dates. For example,

```
strftime("1970-10-25",format="%d%b%y")
```

returns `"25Oct70"`.

The code

```
strftime ("25-12-1970",format="%d%b%y")#
```

results in "19Dec25", and `strftime ("25-12-1970",format="%d%b%Y")` in "19Dec0025".

Time zones and Daylight Saving Time (DST) complicate operations on dates and times. In most demographic applications, time zones and DST are not important. But suppose you want to attend a seminar (webinar) at your institute in Rostock, Germany while visiting San Francisco. The presentation is on 21th June 2013 at 3:00 p.m.; when should you log in? The time is[4]

[4] For more information on this subject, see Revolutions Blog, a blog dedicated to news and information of interest to R users. The editor is David Smith (http://blog.revolutionanalytics.com/2009/06/converting-time-zones.html) (Accessed January 16, 2013).

```
meeting <- as.POSIXct( "2013-06-21 15:00",
                              tz="Europe/Berlin")
```

It is the Central European Summer Time (CEST). The following code produces the time you need to log in:

```
login <- format(meeting, tz="America/Los_Angeles",
                       usetz=TRUE)
```

The result is `"2013-06-21 06:00:00 PDT"`, which is 6:00 a.m. Pacific Daylight Time. An alternative function that gives the same result is the *lubridate* function `with_tz`:

```
with_tz(meeting, "America/Los_Angeles").
```

4. Date as elapsed time since reference point in time

In this section, I consider two formats of elapsed time: (a) Century Month Code, which is the number of months since a fixed reference date and (b) decimal date, which expresses a calendar date as calendar year and fraction of a year. Both formats give a numeric object. Many surveys in the social and health sciences express dates in months since a fixed reference date, without information on the day. The reference date is often January 1, 1900, in which case the coding is referred to as *Century Month Code (CMC)*. CMCs are used in fertility surveys (e.g. the US National Survey on Family Growth), labour force surveys, demographic and health surveys (e.g. Demographic and Health Survey)[5]. The coding scheme is particularly relevant for demographic analysis since it has been used in several surveys, including the World Fertility Survey, launched in the 1970s under the auspices of the International Statistical Institute (ISI). The scheme is documented in demographic texts (see e.g. Pullum 2004). Blossfeld and Rohwer (2002) and Blossfeld et al. (2007) use it in a basic text on event history modeling. The CMC measures the months elapsed since January 1, 1900. For instance, CMC 555 is March 1946 and CMC 1347 is March 2012. The CMC is generally an integer but may be a real number (number with a fractional or decimal component). If the date is known precisely (day, month and year), the CMC is a real number. If the date is known approximately

[5] For the Demographic and Health Survey, see the online guide at http://legacy.measuredhs.com/help/datasets/ (Accessed January 16, 2013) and for the US National Survey on Family Growth, see http://www.cdc.gov/nchs/data/nsfg/NSFG_2006-2010_UserGuide_MainText.pdf (Accessed January 16, 2013).

(month), the CMC is an integer number. If CMC is an integer, the transition is assumed to take place at a given day of the month, usually the beginning of the month or the middle of the month. It is important to make the day explicit since surveys may assume that events occur at the beginning of the month, but that censoring occurs at the end of the month (e.g. the German Life History Survey [GLHS] data distributed by Blossfeld and Rohwer 2002). The CMC coding scheme is applied because the conversion of dates into numerical values facilitates the computation of durations between two dates. A disadvantage is that the interpretation of event dates is much harder.

In demographic analysis, it may be necessary to convert calendar dates into CMC and vice versa. Let us convert January 10, 1984. The `Date_as_cmc` function, which is part of the utilities of Version 2 of the *Biograph* package, released in 2012 (Willekens 2012), converts a calendar date into CMC. For example, `Date_as_cmc("1984-01-10")` gives the CMC 1009 and `Date_as_cmc(c("1984-01-10","1946-03-05"))` results in the vector with elements 1009 and 555[6].

To convert a date in CMC into a calendar date, the components of the date are considered: year, month and day. Year and month are derived from the date in CMC and the day needs to be provided. The *Biograph* function `cmc_as_Date` converts CMC into the calendar date and produces a character string. For example, `cmc_as_Date (1009,15)` is January 15, 1984 and `cmc_as_Date (555)` is March 1, 1946. The first argument is the date in CMC and the second argument is the day. If the day is omitted, it is assumed to be 1 (default). The desired date format may be different, e.g. `cmc_as_Date (555,1,"%d%b%y")` gives us `"01Mar46".`

The decimal date represents a date as a year and fraction of a year. The function `Date_as_year,` which is a utility in *Biograph*, computes the decimal date from the calendar date. It uses relative time; it accounts for the different lengths of months and leap years. For instance,

```
Date_as_year (c("2012-02-29","2010-02-29"),
                              format.in="%Y-%m-%d")
```

converts the first date but not the second date because 2010 is not a leap year and February 29 does not exist. It uses the difftime function of Base R. The result is the numeric vector {2012.161 NA}[7]. Other packages have a similar function. For instance, the function `cal.yr` of the *Epi* package for statistical analysis in

[6] The date functions of Biograph are documented in the manual of the package. The package includes a Doc folder with an R programme that illustrates calls to the different date functions of the package. To find the location of the Doc folder after the package is installed, use `system.file (package="Biograph")`

[7] The common formula that considers the average duration of a month is (see e.g. Mamun, 2003): `year.frac <- year+(month-1)/12+(day-1)/(30.437*12)` where 30.437 is the average number of days in a month.

epidemiology (Carstensen 2012) converts dates into decimal years by assuming that all years are 365.25 days long. To show that the procedure may lead to inaccuracies, Carstensen (2012) converts January 1, 2000, which is 1999.9986. `Date_as_year ("2000-01-01")` gives us 2000. The `decimal_date` function of the *lubridate* package is not able to convert January 1, 2000 to a decimal year. The call `decimal_date (as.Date("2000-01-01"))` results in NA.

To check the precision of the conversion into decimal years, the decimal year may be converted back into a calendar date. The function `year_as_Date` of *Biograph* converts a decimal year into a calendar date. For example, `Date_as_year` converts January 30, 2000 into 2000.079235 , and `year_as_Date` converts 2000.079235 into January 30, 2000. The `year_as_Date` function gives the date in a format specified by the user. If no format is specified, the standard format is used: year, month and day. The code `year_as_Date (2012.161)` gives us February 28, 2012 instead of February 29, 2012. Inaccuracies remain but the reasons are not yet known.

The function `cmc_as_year` converts CMC into decimal date. It uses relative time. Consider March 2012: `cmc_as_year(1347)` is 2012.164 . The decimal date may be converted into CMC using the `year_as_cmc` function of *Biograph*. The function produces an object with two components: the CMC and the day used in converting Date into CMC.

To convert a duration in days to a date, the following code may be used. Suppose a period of 10 days starts on February 20, 2012. The end of the period is

```
as.Date(as.numeric(as.Date("2012-02-20"))+10,
                        origin="1970-01-01")
```

which is March 1, 2012. The procedure accounts for the leap day in the Gregorian calendar. If 2012 is replaced with 2013, the result is March 2, 2013. The operation converts the calendar date (Date object) into a Julian date, adds 10 and converts the Julian date back to a calendar date.

The same result is obtained using a function of *lubridate* package:

```
as.Date("2012-02-20")+days(10).
```

5. Date as elapsed time since reference event

In some observational studies, the date is represented as time elapsed since a reference event. Common reference events are birth, marriage, entry into the job market or an observational study. Common examples of dates as elapsed times since a reference event are age, duration of marriage, number of days since a diagnosis or surgery, and

time-on-study. For instance, the Framingham Heart Study (FHS) gives event dates in days since exam 1 (in addition to SAS dates)[8]. If the calendar date of the reference event is known, ages and durations can be converted into calendar dates. Time measurement as days or years elapsed since a reference event also arises in simulation of life histories. In that case, age is used as the time scale and the number of persons of the same age can easily be determined. To determine the number of individuals of a given age at a given calendar date, dates of birth must be given. If virtual subjects in a simulation are allocated dates of birth, ages can be converted into calendar dates and the size and age composition of the virtual population at a given point in time can be determined.

Age calculations are not trivial because the lengths of years and months vary with their position in the calendar. A division of the number of days since birth by 365.25 is an approximation. In Section 2 of the paper, it became evident that the computation of the exact age at a given date requires an object of class interval. It requires the date of birth and the given date, or the date of birth and the age in days. The `Date_as_age` function of *Biograph* uses *lubridate* functions to define interval and period objects and to compute the exact age. Suppose a person born on July 12, 1977 marries on July 12, 2007, i.e. on his 30th birthday. The following code produces the age:

```
age <- Date_as_age (x="2007-07-12",
                    format.in="%Y-%m-%d",
                    born="1977-07-12").
```

The result is an object with four components: (1) the age in seconds, (2) the age in days, (3) age in years, months and days, and (4) the age in decimal years (years and fraction of a year). The age in decimal years is obtained by subtracting the decimal date of marriage from the decimal date of birth. The function uses *lubridate* functions. Since the computation of age is essential in demographic analysis and the `Date_as_age` function is new, the procedure is described in some detail. The date of birth and date of marriage are:

```
marriage <- ISOdate(2007,07,12,0,0,0)
birth <-  ISOdate (1977,07,12,0,0,0).
```

The following code, using *lubridate* functions, produces the exact age

```
f=new_interval(birth,marriage)
z=as.period (f,unit="year")
```

[8] http://www.framinghamheartstudy.org/share/data/soe_06as.html (Accessed January 16, 2013)

The result is "30y 0m 0d 0H 0M 0S", as it should be. The age in completed years is `z@year`. An alternative way of obtaining the age is `f%/%years(1)`.

The time difference between marriage and date of birth is 10957 days. The age at marriage in days is obtained by

```
age.d <- new_interval(ymd("1977-07-12"),
                      ymd("2007-07-12")) %/% days(1)
```

or

```
difftime(ymd("2007-07-12"),ymd("1977-07-12")).
```

The age in seconds is obtained by converting the interval object into a duration object:

```
d <- as.duration (f)
```

with d an object of class “duration”. This gives us 946684800 seconds, which in that period is exactly 30.00 years. The number of days is the number of seconds divided by the number of seconds per day, which is 86400. It is 946684800 /86400 = 10957 days.

If a year is assumed to have 365.25 days, the age is 10957/365.25 = 29.9986 years or 29 years in completed years. Division should be by years of relative length and not by years of constant length.

The date of birth and the age may be converted into a date using *Biograph*’s `age_as_Date` function:

```
age_as_Date (30,"1977-07-12",format.born="%Y-%m-%d",
                             format.out="%Y-%m-%d").
```

The date is "2007-07-12".

6. Illustrative applications

Calot wanted to know the effect of the number of Saturdays in a month on the monthly marriage rate. The estimation of the Saturday effect on monthly marriage rates is difficult to detect in real data because of relatively strong seasonal effects (see e.g. Haskey 1996; Wilson and Smallwood 2007). Therefore a hypothetical example was given in the introduction. I am not aware of recent studies that identify the Saturday effect on the monthly marriage rate. A comparable issue is the effect of number of Sundays in a month on the number of births. Fewer children are born on Sunday (and

Saturday) than on a weekday. The daily distribution of births changed dramatically, starting in the late 1960s (Kotzamanis and Kostaki 2011). Kotzamanis and Kostaki (2011) calculate coefficients to estimate the day-of-the-week effect and the Vienna Institute of Demography does the same as part of "Geburtenbarometer". In the USA during the period 2005-2010, an average of 7400 children were born on a Sunday, 8500 on a Saturday and 13300 on a Tuesday. Consider 2008. Months with five Sundays had an average of 7335 fewer births (2 percent) than months with four Sundays and the Sunday effect is significant at the 1 percent level. The General Fertility Rate (number of births per 1000 women aged 15 to 44, on an annual basis for specified month), declines from 69 in months with four Sundays to 67.5 in months with five Sundays. The decline is larger if seasonally adjusted birth rates are used. In months with five Sundays, the fertility level is 2.5 children per 1000 women aged 15 to 44 below that in months with four Sundays and the Sunday effect is significant at a 1 percent level.

The Sunday effect is determined in five steps. The first step is to list all the dates between January 1, 2008 and December 31, 2008:

```
start <- as.Date("2008-01-01")
stop <- as.Date("2008-12-31")
d <- seq(start,stop,by=1).
```

The second step is to select the dates that are Sunday:

```
dd = subset (d,weekdays(d)=="Sunday").
```

In the third step, Century Month Codes (CMC) are generated for the months in the vector `dd`. Using CMCs is a simple way to differentiate between months in successive years.

```
cmc <- as.POSIXlt(dd)$year*12+as.POSIXlt(dd)$mon+1
```

The object `cmc` is numeric. The code `table(cmc)` gives the number of Sundays per month in 2008. In the fourth step, a dummy variable is generated for regression analysis. The variable is 1 for months with 5 Sundays and 0 otherwise:

```
cmc.index <- ifelse (table(cmc)==5,1,0).
```

In the final step, a Poisson regression model is used to determine the Sunday effect.

The second application is episode-splitting. In survival analysis, the presence of a time-varying covariate often requires that an episode is divided in two episodes, one

before the change in the time-varying covariate and one after the change[9]. For instance, Blossfeld and Rohwer (2002) wanted to know how entry into first marriage impacts the job-exit rate. Job exit is the event of interest and marital status is a time-varying covariate. A job episode in which a person marries is split in two episodes at the date of marriage. In survival and event history analysis, episode-splitting is a technique to study the effect of time-varying covariates (see also Mills 2011).

Suppose a person enters a job on September 20, 2011, marries on Saturday, May 5, 2012 and leaves the job for another job on July 2, 2012. From the job entry date to the marriage date or censoring, the person contributes exposure time to the group of non-married persons. From marriage to the job exit date or censoring, the person contributes exposure time to the group of married persons. The following code splits the job episode in two episodes (before marriage and after marriage) and creates two records. It splits the job episode that starts on September 20, 2011 and ends on July 2, 2012 at the date of marriage (May 5, 2012). The code is:

```
start <- as.Date ("2011-09-20")
marriage <- as.Date ("2012-05-05")
stop <- as.Date("2012-07-02")
d <- seq(start,stop,by=1)
m.group <- ifelse (d<marriage,0,1)
data <-data.frame(ID=10,Start=start,Marriage=marriage,
  Stop=stop)
newdata <- lapply(data, rep, 2)
newdata <- do.call("data.frame", newdata)
newdata$ms <- c(0,1)
start2 <- ifelse (newdata$ms==1,marriage,start)
stop2  <- ifelse (newdata$ms==0,marriage,stop)
newdata$Start <- as.Date(start2,origin="1970-01-01")
newdata$Stop <- as.Date(stop2,origin="1970-01-01").
```

The `ifelse` function produces objects `start2` and `stop2` that have the same attributes, including class, as the test object (which is numeric). Therefore, `start2` and `stop2` are numeric. Their values are Unix dates of job entry, job exit, and marriage (number of days since January 1, 1970). The numeric objects need to be converted into "Date" objects. The code adopts essential elements of the `survSplit` function of the *survival* package (Therneau and Lumley 2012), which splits a record of a survival object (an episode) in multiple records.

[9] Another approach is to specify a multistate model (Aalen et al. 2008, p. 358).

The third application is the simulation of population change produced by changes in longevity and life histories. When time is measured in integral months and simulation proceeds month by month, such as in SOCSIM (Wachter et al. 1997: Mason 2011), differences in length of months are disregarded. When time is continuous and a duration is expressed in days or seconds, a calendar is required to convert days (seconds) into months and years. In LifePaths of Statistics Canada (Gribble et al. 2003) and MicMac (Zinn and Gampe 2011), events can occur at any arbitrary moment, and are not artificially restricted to annual or monthly intervals. Simulations take place in continuous time. LifePaths is part of Modgen, a generic microsimulation programming language, developed at Statistics Canada[10]. In this section I describe the time tracking in MicMac. The MicMac source code can be downloaded from the website of the Max Planck Institute for Demographic Research[11]. MicMac consists of a multistate cohort-survival population projection model (Mac) and a microsimulation model (Mic). A pre-processor reads the data and estimates transition rates, a simulation engine performs the microsimulations (MicCore), and a post-processor tabulates and graphs the results. The pre- and post-processors are in R and the simulation engine is in Java. The microsimulation model generates individual life histories from transition rates between functional states, birth rates and death rates. The rates are age-specific and they may vary by covariate and change in time. They are the parameters of waiting time distributions that are piecewise exponential. From these distributions random waiting times are drawn to determine who moves between states, what transitions occur, and when they occur. Simulation starts at a given date (e.g. January 1, 2004) and ends at another date (e.g. December 31, 2050). The population at the starting date is the initial population. Between the two dates, individuals transfer between states, children are born, and some individuals die. Individuals may also immigrate and emigrate. Ages at transition are measured in milliseconds. The exact dates of birth of the starting population are not known, but the years of birth are known because at the starting date the age of the population is given in single years of age. Dates of birth are assigned randomly to a day of the year by drawing from the standard uniform distribution U[0,1] (Zinn and Gampe 2011, p. 28). Time at birth is measured in elapsed time: milliseconds since midnight January 1, 1970. The number is negative for births before that date. Milliseconds are converted into days using the formula: s/1000/60/60/24, where s is the number of milliseconds since the reference time. Event times are also converted into calendar dates (format: "YYYY-mm-dd"). The calendar date of an event is obtained as the sum of the reference date (January 1, 1970), an object of class “Date”, and the elapsed time in days. The result is an object of class “Date”. Consider a virtual individual aged 20 on January 1, 2004. The individual was born in 1983. Suppose the

[10] http://www.statcan.gc.ca/microsimulation/index-eng.htm (Accessed January 16, 2013).
[11] http://www.demogr.mpg.de/cgi-bin/micmac/login.plx (Accessed January 16, 2013).

random draw results in the individual being born at midnight on August 25, 1983. The time of birth is 430.6 million seconds (0.4306176e+12 milliseconds) or 4984 days since the reference time.

```
d=ISOdate (1983,08,25,0,0,0)
milliseconds = as.numeric (as.Date(d)) *86400 *1000
```

The virtual individual enters 'observation' (simulation) at midnight on January 1, 2004 (1.072915e+12 elapsed milliseconds), gets a first child at midnight November 18, 2032 (1.984349e+12 elapsed milliseconds) and a second child on September 12, 2034. The age at birth of the first child and the interval between the first and the second child are outcomes of randomly drawing waiting times from distributions that describe time to the next child, time to a change in marital status and time to death. The shortest waiting time determines the transition that occurs and the age at transition. Immigrants are generated from immigration rates by age and covariates. One female immigrant in the virtual population is born on March 1, 2012, immigrates on November 30, 2013, and has three children. They are born on October 20, 2031, October 7, 2037 and October 6, 2045. The observation is censored on December 31, 2050.

The date functions of MicMac are in the pre- and postprocessor `micmacPreFun.r` and `micPostFun.r`. The random draws and the computation of dates are done in the function `dateTransformation`, which transforms birth dates and transition times into dates. The ages at transition and the durations between transitions are computed in the function `diffBetweenDates`, which computes the difference in years between two dates. The function accounts for leap years. MicMac computes the age on January 1, 2005 of the individual born on August 25, 2003 as follows:

```
cal1.year - cal2.year - 1 + daysInSimYear +
                                restDaysInBirthYear
```

where `cal1.year=2005, cal2.year=2003`, `restDaysInBirthYea`r is the number of days spent in 2003 between the date of birth and the end of the year (130 days) divided by 365, and `daysInSimYear` is the number of days spent in 2005 (1 day) divided by 365. On January 1, 2005 (end of that day), the person is 1.36 years of age (decimal year). This illustrates operations on dates in microsimulation in continuous time. The dates and ages may also be obtained using the functions discussed earlier in this paper.

7. Conclusion

Calendar time (period, cohort) and age are key concepts in demography. The computation of age and duration is central to the discipline. Age calculations generally involve an approximation, although a good one. The approximation uses months and years of constant length, whereas in reality the duration of a month or a year depends on when the month or year occurs, i.e. on the position in the calendar. In recent years there has been a growing interest in arithmetic and logical operations on dates. This paper introduces a systematic treatment of chronological objects in demography. Two representations of dates are distinguished. The first is the Gregorian calendar, which represents dates by year, month and day. The second is the Julian date, which represents dates as the number of days since a reference point in time (epoch), and its extensions: (a) months since a reference point in time and (b) days since a reference event. These two categories encompass all representations of dates in data collection (surveys, follow-up studies, etc.) and data processing. Operations on dates require a conversion of one date representation into another. A number of conversion algorithms and the associated R functions are presented in this paper. The availability of conversion algorithms reduces the need for date coding schemes, such as the Century Month Code. Century Month Codes facilitate calculations of durations but hinder the interpretation of raw data and exploratory data analysis.

As the technology evolves and conversion algorithms become widespread in software packages and spreadsheets, arithmetic and logical operations on chronological objects are expected to increase. Conversions between calendars, e.g. the Chinese, Hindu, Islamic and Gregorian calendars, are expected to become widely accessible, resulting in opportunities for demographic studies within and across calendar systems. Conversion algorithms will change data collection practices because they reduce the need for conversion of date of birth and dates at other demographic events from the local calendar (e.g. lunar calendar) to the Gregorian calendar or another standard. The development is expected to increase the accuracy of date and age reporting. A systematic investigation of chronological objects may also address the cross-cultural differences in counting a person's age. In the East Asian system of age reckoning, which originated in China, a newborn starts at age one and each Lunar New Year, rather than the birthday, adds one year to the person's age. As a result, people may be one or two years older in the Asian system of age reckoning than in the Western system, which makes age reporting ambiguous (Zeng and Gu 2009, p. 62). A systematic study of chronological objects will gradually remedy the many problems date and time measures pose in demographic research. A good starting point of a systematic study is the distinction between instant, interval, period and duration, recently introduced in R by Grolemund and Wickham (2011).

8. Acknowledgements

I thank the referees for their excellent comments and suggestions. I also thank Maria Winkler-Dworak for the information on the use of the day-of-the-week effect in the "Geburtenbarometer".

References

Aalen, O.O., Borgan, Ø., and Gjessing, H.F. (2008). *Survival and event history analysis. A process point of view.* New York: Springer.

Blossfeld, H.P., Golsh, K., and Rohwer, G. (2007). *Event history analysis with Stata*. Mahwah, NJ: Erlbaum.

Blossfeld, H.P. and Rohwer, G. (2002). *Techniques of event history modeling. New approaches to causal analysis.* Mahwah, NJ: Erlbaum

Calot, G. (1981). Le mouvement journalier des naissances à l'intérieur de la semaine (Daily movement of births during the week). *Population* 36(3): 477-504. doi:10.2307/1532617.

Calot, G. and Sardon, J.P. (2004). Methodology for the calculation of Eurostat's demographic indicators. Luxembourg: Eurostat, European Demographic Observatory. (Population and social conditions; 3/2003/F/no 26)

Carstensen, B. (2012) Epi: A package for statistical analysis in epidemiology. Version 1.1.40. http://cran.r-project.org/web/packages/Epi/index.html (accessed January 16, 2013).

Chalabi, Y., Mächler, M., and Wuertz, D. (2011). Rmetrics – timeDate package. *The R Journal* 3(1): 19-24.

Colebourne, S. and O'Neill, B.S. (2010). Joda-Time - Java Date and Time API. http://joda-time.sourceforge.net (accessed January 16, 2013).

Doblhammer, G. and Vaupel, J.W. (2001). Lifespan depends on month of birth. *Proceedings of the National Academy of Sciences of the United States* 98(5): 2934-2939. doi:10.1073/pnas.041431898.

Freemantle, N., Richardson, M., Wood, J., Ray, D., Khosla, S., Shahian, D., Roche, W.R., Stephens, I., Keogh, B., and Pagano, D. (2012). Weekend hospitalization and additional risk of death: An analysis of inpatient data. *Journal of the Royal Society of Medicine* 105(2): 74-84. doi:10.1258/jrsm.2012.120009.

Goodman, M.J., Nelson, W.W., and Maciosek, M.V. (2005). Births by day of week: A historical perspective. *Journal of Midwifery and Women's Health* 50(1): 39-43. doi:10.1016/j.jmwh.2004.09.005.

Gray, B.J. and Lumley, T. (2010) ConvCalendar: converts dates between calendars. Version 1.1. http://cran.r-project.org/web/packages/ConvCalendar/index.html (Accessed January 16, 2013).

Gribble, S., Hicks, C., and Rowe, G. (2003). *The LifePaths microsimulation model.* Paper presented at the International Microsimulation Conference on Population, Ageing and Health: Modelling Our Future. Canberra, Australia: University of Canberra, NATSEM.

Grolemund, G. and Wickham, H. (2011). Dates and times made easy with lubridate. *Journal of Statistical Software* 40(3).

Grolemund, G. and Wickham, H. (2012). Package lubridate (Version 1.2.0). http://cran.r-project.org/web/packages/lubridate/index.html (Accessed January 16, 2013).

Haskey, J. (1996). The day of the week on which couples marry. *Population Trends* 85: 45-52.

Hong, J.S., Kang, H.C., Yi, S-W., Han, Y.J., Nam, C.M., Gombojav, B. and Ohrr, H. (2006). A comparison of perinatal mortality in Korea on holidays and working days. *BJOG: An International Journal of Obstetrics & Gynaecology*. 113(11): 1235-1238. doi:10.1111/j.1471-0528.2006.01054.x.

IBM (2011). *IBM SPSS Statistics 20 Core System User's Guide*. Chicago: IBM Software Group.

James, D. and Hornik, K. (2011). chron: Chronological objects which can handle dates and times. R package (Version 2.3-43), http://cran.r-project.org/web/packages/chron/index.html (Accessed January 16, 2013).

James, D.A. and Pregibon, D. (1993). Chronological objects for data analysis. In: Tarter, M.E. and Lock, M.D. (eds.). *Statistical applications of expanding computer capabilities: Proceedings of the 25th Symposium on the Interface*. Fairfax Station, VA: Interface Foundation of North America: 177-182.

Kalbfleisch, J.D. and Prentice, R.L. (2002). *The statistical analysis of failure time data*. New York: Wiley.

Kotzamanis, B. and Kostaki, A. (2011). *Seasonality of births in Europe and the USA: A comparative approach*. Paper presented at the Annual Meeting of the Population Association of America, Washington D.C., March 31 – April 2, 2011.

Lerchl, A. and Reinhard, S.C. (2008). Where are the Sunday babies? II. Declining weekend birth rates in Switzerland. *Naturwissenschaften* 95(2): 161-164 doi:10.1007/s00114-007-0305-4.

Mäkelä, P., Martikainen, P., and Nihtilä, E. (2005). Temporal variation in deaths related to alcohol intoxication and drinking. *International Journal of Epidemiology*, 34(4): 765-771. doi:10.1093/ije/dyi025.

Mamun, A.A. (2003). *Life history of cardiovascular disease and its risk factors: multistate life table approach and application to the Framingham Heart Study.* Amsterdam: Rozenberg Publishers.

Mason, C. (2011). Socsim oversimplified. Berkeley: Demography Lab, University of California. http://lab.demog.berkeley.edu/socsim/socsimOversimplified.pdf (Accessed January 16, 2013).

Mills, M. (2011). *Introducing survival and event history analysis.* London: Sage Publications.

Mohammed, M.A., Sidhu, K.S., Rudge, G., and Stevens, A.J. (2012). Weekend admission to hospital has a higher risk of death in the elective setting than in the emergency setting: A retrospective database study of national health service hospitals in England. *BMC Health Services Research* 12(87). doi:10.1186/1472-6963-12-87.

Mooney, J.A., Helms, P.J., and Jolliffe, I.T. (2004). Higher incidence of SIDS at weekends, especially in younger infants. *Archives of Disease in Childhood* 89(7): 670-672. doi:10.1136/adc.2002.023408.

NHS-London (2011). Adult emergency services: Acute medicine and emergency general surgery. Case for change. London: National Health Service. http://www.londonhp.nhs.uk/wp-content/uploads/2011/09/AES-Case-for-change_September-2011.pdf (Accessed January 16, 2013).

Pasupathy, D., Wood, A.M., Pell, J.P., Fleming, M., and Smith, G.C.S. (2010). Time of birth and risk of neonatal death at term: Retrospective cohort study. *British Medical Journal* 341(c3498). doi:10.1136/bmj.c3498.

Phillips, D., Barker, G.E., and Brewer, K.M. (2010). Christmas and New Year as risk factors for death. *Social Science and Medicine* 71(8): 1463-1471. doi:10.1016/j.socscimed.2010.07.024.

Pullum, T.W. (2004). Natality. Measures based on censuses and surveys. In: Siegel, J. and Swanson, D. (eds.). *The Methods and materials of demography.* Amsterdam: Elsevier: 407-428.

Salihu, H.M., Ibrahimou, B., August, E.M., and Dagne, G. (2012). Risk of infant mortality with weekend versus weekday births: A population-based study. *The*

Journal of Obstetrics and Gynaecology Research 38(7): 973-979. doi:10.1111/j.1447-0756.2011.01818.x.

SAS Institute (2008). SAS/STAT 9.2 User's guide. The PHREG procedure (Book Excerpt). Cary, NC: SAS Institute.

Sobotka, T., Winkler-Dworak, M., Testa, M.R., Lutz, W., Philipov, D., Engelhardt, H., and Gisser, R. (2005). Monthly estimates of the quantum of fertility: Towards a fertility monitoring system in Austria. *Vienna Yearbook of Population Research* 3: 109-141.

St.Clair, P., Bugliari, D., Campbell, N., Chien, S., Hayden, O., Hurd, M., Main, R., Miu, A., Moldoff, M., Panis, C., Pantoja, P., Rastegar, A., Rohwedder, S., Oshiro, M., and Zissimopoulos, J. (2011). RAND HRS data documentation, Version L. Santa Monica, CA: RAND Center for the Study of Aging. http://hrsonline.isr.umich.edu/modules/meta/rand/randhrsl/randhrsL.pdf (Accessed January 16, 2013).

Stata (2009). *Stata survival analysis and epidemiological tables reference manual. Release 11*. College Station. TX: StataCorp.

Therneau, T.M. and Grambsch, P.M. (2000). *Modeling survival data. Extending the Cox model.* New York: Springer.

Therneau, T.M. and Lumley, T. (2012). survival: Survival analysis, including penalised likelihood. R package. (Version 2.37-2). http://cran.r-project.org/web/packages/survival/index.html (Accessed January 16, 2013).

Therneau, T.M., Lumley, T., Halvorsen, K. and Hornik, K. (2012). date: Functions for handling dates. R package. (Version 1.2-33). http://cran.r-project.org/web/packages/date/index.html (Accessed January 16, 2013).

Wachter, K.W., Blackwell, D., and Hammel, E.A. (1997). Testing the validity of kinship microsimulation. *Mathematical and Computer Modeling*. 26(6): 89-104. doi:10.1016/S0895-7177(97)00172-6.

Willekens, F. (2012). Biograph: Explore life histories. R package. (Version 2.0.2). http://cran.r-project.org/web/packages/Biograph/index.html (Accessed January 16, 2013).

Wilson, B. and Smallwood, S. (2007). Understanding recent trends in marriage. *Population Trends* 128: 24-32.

Wuertz, D., Chalabi, Y., Maechler, M. et al. (2012). timeDate: Rmetrics– Chronological and calendar objects. (Version 2160.97). http://cran.r-project.org/web/packages/timeDate/index.html (Accessed January 16, 2013).

Zeng, Y. and Gu, D. (2009). Reliability of age reporting among Chinese oldest-old in the CLHLS datasets. In: Zeng, Y., Poston, D.L., Vlosky, D.A., and Gu, D. (eds.). *Healthy longevity in China. Demographic, socioeconomic and psychological dimensions*. New York: Springer: 61-78.

Zinn, S. and Gampe, J. (2011). The MicMacCore manual. Rostock: Max Planck Institute for Demographic Research (MPIDR).

Annex

Base R and a number of packages in the Comprehensive R Archive Network (CRAN) (http://cran.r-project.org/) offer useful R functions for operations on dates. In this paper, date functions of Base R and of the packages *lubridate* and *Biograph* are used. The package *lubridate* (Grolemund and Wickham 2011, 2012) contains functions to identify and parse date-time data, extract and modify components of a date-time (years, months, days, hours, minutes, and seconds), perform accurate math on date-times, handle time zones and Daylight Saving Time. A particularly useful feature is the distinction between time units of fixed length and time units of variable length, and the associated four time-related objects: instant, interval, duration and period. *Biograph* (Willekens 2012) is designed to facilitate the descriptive and statistical analysis of life histories. It follows a multistate perspective on the life course and conceptualizes the life course as a sequence of states and transitions between states (events). Transitions are governed by transition rates estimated from transition data. Transition rates vary with age and may depend on covariates.

Table A shows date functions used in this paper. For each function the following information is provided: name, package that includes the function, main task, and an example.

Table A: Functions in R for operations on dates (used in the paper)

Function	Package	Task	Example
as.Date	Base R	Converts character or numeric representations of dates into dates	`as.Date("2010-02-25")` `as.Date(14665,origin="1970-01-01")`
ISOdate	Base R	Creates date-times from numeric representations	`ISOdate (2012,02,25,0,0,0, tz="UTC")`
as.POSIXtl	Base R	Converts objects into dates and manipulates the objects	`as.POSIXlt ("2010-02-25 00:00:00")`
strptime strftime	Base R	Converts between character representations of dates and objects of classes POSIXlt and POSIXct	`strptime("25feb2010","%d%b%Y")`
difftime	Base R	Computes length of time interval between two dates	`difftime(as.Date("2007-07-12"), as.Date("1977-07-12"))`
format	Base R	Converts between character representations of dates and objects of class "Date"	`format ( as.POSIXct("2013-06-20 15:00", tz="Europe/Berlin"), tz="America/New_York",usetz=TRUE)`

Table A: (Continued)

Function	Package	Task	Example
Date_as_cmc	Biograph	Converts date object into Century Month Code	`Date_as_cmc( as.Date("2010-02-25"))`
Date_as_year	Biograph	Converts date object into decimal year	`Date_as_year( as.Date("2010-02-25"))`
Date_as_age	Biograph	Converts date into age, given date of birth	`Date_as_age ("2010-02-25", format.in="%Y-%m-%d",born= "1983-07-17")`
age_as_Date	Biograph	Converts age into calendar date, given date of birth	`age_as_Date (30, born="1983-03-17", format.born="%Y-%m-%d", format.out="%d-%b-%y")`
age_as_year	Biograph	Converts age into decimal year, given date of birth	`age_as_year(30,born="1983-07-17",format.born="%Y-%m-%d")`
cmc_as_Date	Biograph	Converts Century Month Code into calendar date	`cmc_as_Date (x=555,selectday=1)`
cmc_as_year	Biograph	Converts Century Month Code into decimal year	`cmc_as_year (1322)`
year_as_cmc	Biograph	Converts decimal year into Century Month Code	`year_as_cmc(2010.084932)`
year_as_Date	Biograph	Converts decimal year into calendar date	`year_as_Date (x=2010.150685, format.out='%d-%m-%Y')`
dmy	lubridate	Parses dates according to the order that year, month, and day elements appear in the input	`dmy (17022010)`
new_interval	lubridate	Creates interval object with specified start date and end date	`new_interval ( as.Date ("2009-01-30"), as.Date("2012-02-25"))`
as.period	lubridate	Changes interval, duration, difftime and numeric class objects to period class objects with the specified units	`span <- new_interval( as.Date("2009-01-30"), as.Date("2012-02-25")) as.period (span,unit="year")`

Table A: (Continued)

Function	Package	Task	Example
as.duration	lubridate	Converts interval object into duration object	`f=new_interval ( as.Date ("2009-01-30"), as.Date("2012-02-25")) as.duration (f)`
diffBetweenDates	MicMac	Computes difference between dates in years	`diffBetweenDates ("2010-02-25", "1990-07-17")`
LeapYear	MicMac	Determines whether a year is a leap year (TRUE / FALSE)	`LeapYear (2008)`
dateTransformation	MicMac	Transforms birth times and transition times into calendar dates	*Designed for MicMac input date file*