The "rtv" Package
Random Time Variable Objects

Charlotte Maia

February 8, 2009

Abstract

The rtv package is an R package for conveniently representing and manipulating realisations of random time variables. Common examples include reading formatted time strings (e.g. "2008-01-01 06:00:00") and converting them to a continuous measure (or vice versa), and applying mathematical operations to realisations (e.g. the mean realisation). An object oriented paradigm is used, where realisations are represented by rtv objects. These can either be drtv objects (corresponding to values from a calendar and a clock, which are regarded as a discrete representation of time) or crtv objects (corresponding to values from a real number line augmented by origin and unit attributes, which are regarded as a continuous representation of time). In the continuous case the unit can be either year, month, day, hour, minute, second or week. Special consideration has been given to using years and months as units.

Warning

This is the first version of this package. The package is likely to have at least moderate changes in subsequent versions, plus has had minimal testing. At present no testing has been done for dates hundreds of years into the past or future. Anyone using this package, should use caution, testing each step of output carefully.

1 Introduction

The rtv package is a package for working with realisations of random time variables. Roughly speaking we can regard this as a set of tools for working with time data, not unlike R POSIXlt and POSIXct objects (which are actually used by the rtv package).

Before we can discuss the rtv package in more detail, we must first understand the key principles that underlie it, what exactly we mean by a random time variable, and problems with existing packages.

There are two key sets principles underlying the package:

1. The word time is ambiguous. Not only is it used to mean different things in different contexts, it is sometimes used to mean different things in the same contexts. One possible definition for time, is that time is a random variable (i.e. a random time variable), where the random variable has a special kind of sample space described below. The notion that time is a random variable is very important and many real world processes can (and should) be modelled with random time variables.

2. Time realisations are common in real world datasets, especially in datasets extracted from business and government databases. Often raw data will be expressed as formatted time strings (e.g. "2008-01-01 06:00:00") or in some other format that is difficult to analyse. This means that raw time data

often must first be processed prior to statistical modelling. Depending on the data and the type of model required this can be a very time consuming and very error prone task.

We need to define time in more detail, however it is first necessary to define a time unit event. We define a time unit event to be any of the events {year, month, day, hour, minute, second, week}. A common operation is counting the number of time unit events that occur between two instants. In this vignette it is assumed that such a count is for a single type of time unit event only (e.g. number of days or number of seconds, but not number of days and seconds) and that such a count can be fractional.

The count forms our most basic definition of time, the number of some time unit events that occur between two instants. However, in many situations it is easier to work with a representation based on combining values from a Gregorian calendar and a 24 hour clock. In either case, if we take all possible values that such time can take, then we could describe this set of time values as a time axis or a time sample space.

Now we can define a random time variable as a random variable whose sample space is, intuitively, a time sample space. We can also describe realisations of a random time variable as time realisations.

This package does not represent random time variables directly, but rather realisations of random time variables. However repeating the previous statement, the notion that time is a random variable is very important.

In statistical models, time is usually regarded as a timeseries or as a real number. In the timeseries case, raw data will generally require little processing, as time can generally be replaced by an index {1, 2, . . . , n}. However in the real case, raw data may need to be mapped from time strings to counts (as defined above), or changed from one continuous measure to another (noting that choice of time units can effect an analysis). In both cases it may be necessary to format the output of analysis in order to fully interpret it.

For users of R, the obvious tools are the Date, POSIXlt and POSIXct classes. There are also a number of additional packages for working with timeseries data, however these additional packages generally contradict the notion that time is a random variable, so are not discussed further.

The POSIX time classes in R, are very powerful, and with a little expertise can be used to perform most operations that one is likely to require. However, they are (in my opinion) counter-intuitive and there are also several nuisances including:

1. A lack constructors (necessary for a clean object oriented design). Objects can be created by coercing another object or calling strptime.

2. Sensitivity to timezone. This is generally redundant in statistical modelling and can cause unexpected results.

3. With POSIXct objects, time is expressed as the number of seconds since "1970-01-01 00:00:00", although by default output is formatted. This is neither intuitive nor convenient for mathematical purposes.

4. It is not directly possible to compute the number of years or months that have occurred between two instants (noting that yearly cycles are common in many contexts).

The main goal of the rtv package is to provide a set of tools for working with time realisations, which are intuitive and convenient to a statistician. As mentioned above the POSIX time classes in R are very powerful and hence many parts of the rtv package are built on top of these, although in general, they are hidden from the user.

Additionally, the rtv package has the following goals:

1. Conveniently represent time realisations. This means either a combination of values from a Gregorian calendar and a 24 hour clock (discrete time), or values from a real number line representing the number of time unit events between two instants, augmented by origin and unit attributes (continuous time).

The origin represents the time at which the first instant occurred (on a separate standard time axis) and the unit can be any time unit event as defined above.

2. All time is treated as GMT time. This ensures that (in theory) the identity $x + 1$ day $= x + 24$ hours is always true, for any valid time realisation x.

3. Time realisations should be represented using an object oriented paradigm.

4. There should be a large number of straight forward constructors for creating realisation objects from a variety of seed objects.

5. In general, mathematical operations applied to realisation objects should also return realisation objects.

6. By default time output should not be formatted.

The realisation objects described above are implemented as rtv objects (random time variable objects). An rtv object is either a drtv object (discrete random time variable object) or a crtv object (continuous random time variable object).

The drtv objects are fairly trivial (and have a similar structure to POSIXlt objects). Objects contain a list of eight equal length numeric vectors (seven of which are in principle integers). The first six correspond to year (any integer value), month (1-12), day (1-31), hour (0-23), minute (0-59) and second (0-59). The last two are the day of the week (1:Monday-7:Sunday) and the day of the year (1-366).

The crtv objects are are essentially a numeric vector with scalar origin and unit attributes. The origin attribute is a POSIXct object or any object that can be coerced to a POSIXct object (noting that this might be changed to a crtv object in a later version of the package). The unit attribute is a character whose value is the name of any time unit event.

Not only can rtv objects be created from a variety of other objects, but rtv objects can also be created from other rtv objects. In the special case of creating crtv objects from other crtv objects we can change the origin or the unit. Often this is mathematically trivial. Changing the origin from 2000-01-01 to 2001-01-01 means subtracting 366 days from each realisation. Changing the unit from day to week, means dividing the realisations by seven. However changing the unit from day, hour, minute, second or week to year or month requires special consideration. The approach taken here, is that a a period of one year, corresponds to the exactly the same month-day date one year apart regardless of the number of days involved. The same principle applies to months. This is discussed in more detail in the section on creating crtv objects.

2 Creating and Formatting drtv Objects

Possibly the most practical use of the rtv package is reading formatted time strings. In the following example a character vector of formatted time strings is created, then a drtv object is created using the character vector as a seed object.

```
> seed = c ("2008-01-01", "2008-02-01", "2008-03-01", "2008-04-01")
> x = drtv (seed)
> x
$year
[1] 2008 2008 2008 2008

$month
[1] 1 2 3 4
```

```
$day
[1] 1 1 1 1

$hour
[1] 0 0 0 0

$minute
[1] 0 0 0 0

$second
[1] 0 0 0 0

$dow
[1] 2 5 6 2

$doy
[1]  1 32 61 92

attr(,"class")
[1] "drtv" "rtv"
```

First note the class attribute. A drtv object is also an rtv object. The same principle applies to crtv objects discussed in the next section.

Note that we can produce a formatted version.

```
> explicit.format (x)
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
```

We can also extract individual components.

```
> x$dow
[1] 2 5 6 2
```

This can also be formatted (more on this later).

```
> dow.string (x$dow)
[1] "Tue" "Fri" "Sat" "Tue"
```

We can also have a date-clock format.

```
> seed = c ("2010-01-01 12:15:00", "2010-01-01 12:16:00",
         "2010-01-01 12:17:00", "2010-01-01 12:18:00")
> x = drtv (seed, date=FALSE)
> x
$year
[1] 2010 2010 2010 2010

$month
[1] 1 1 1 1

$day
```

```
[1] 1 1 1 1

$hour
[1] 12 12 12 12

$minute
[1] 15 16 17 18

$second
[1] 0 0 0 0

$dow
[1] 5 5 5 5

$doy
[1] 1 1 1 1

attr(,"class")
[1] "drtv" "rtv"
> explicit.format (x, date=FALSE)
[1] "2010-01-01 12:15:00" "2010-01-01 12:16:00" "2010-01-01 12:17:00"
[4] "2010-01-01 12:18:00"
```

Or a format of our choice, using the same syntax used by strptime (refer to the help file for this function if necessary)

```
> seed = c ("2010:01:01-12:15:00", "2010:01:01-12:16:00",
         "2010:01:01-12:17:00", "2010:01:01-12:18:00")
> tf = "%Y:%m:%d-%H:%M:%OS"
> x = drtv (seed, informat=tf)
> x
$year
[1] 2010 2010 2010 2010

$month
[1] 1 1 1 1

$day
[1] 1 1 1 1

$hour
[1] 12 12 12 12

$minute
[1] 15 16 17 18

$second
[1] 0 0 0 0
```

```
$dow
[1] 5 5 5 5

$doy
[1] 1 1 1 1

attr(,"class")
[1] "drtv" "rtv"
> explicit.format (x, outformat=tf)
[1] "2010:01:01-12:15:00" "2010:01:01-12:16:00" "2010:01:01-12:17:00"
[4] "2010:01:01-12:18:00"
```

The drtv objects can also be created from rtv, Date, POSIXlt and POSIXct objects, as well as offering a default constructor. Refer to the help file for drtv for more information.

Note that if the user does not wish to call explicit.format there is an option to make formatting automatic. The user may also need to set an option which controls whether or not the date only or date-clock form is used (in printing). There are also options to change the default format used for the date only string and the date and clock string.

```
> #change the options
> options (rtv.explicit.format=TRUE)
> options (rtv.print.date=FALSE)
> tf.old = getOption ("rtv.default.format.long")
> #for date only format: options (rtv.default.format.short="...some format...")
> options (rtv.default.format.long="%Y:%m:%d-%H:%M:%OS")


> #equivalent calls to previous
> seed = c ("2010:01:01-12:15:00", "2010:01:01-12:16:00",
         "2010:01:01-12:17:00", "2010:01:01-12:18:00")
> drtv (seed, date=FALSE)
[1] "2010:01:01-12:15:00" "2010:01:01-12:16:00" "2010:01:01-12:17:00"
[4] "2010:01:01-12:18:00"


> #change things back
> options (rtv.explicit.format=FALSE)
> options (rtv.print.date=TRUE)
> options (rtv.default.format.long=tf.old)
```

3 Creating and Interconverting crtv Objects

We can create and format crtv objects in an almost identical way to drtv objects.

```
> seed = c ("2008-01-01", "2008-02-01", "2008-03-01", "2008-04-01")
> x = crtv (seed)
> x
```

```
[1] 2922 2953 2982 3013
attr(,"class")
[1] "crtv" "rtv"
attr(,"origin")
[1] "2000-01-01 GMT"
attr(,"unit")
[1] "day"
> explicit.format (x)
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
```

The key difference is that we can specify origin and unit attributes (the defaults are 2000-01-01 00:00:00 and day).

```
> seed = c ("2008-01-01", "2008-01-08", "2008-01-15", "2008-01-22")
> crtv (seed, origin=crtv ("2008-01-01"), unit="week")
[1] 0 1 2 3
attr(,"class")
[1] "crtv" "rtv"
attr(,"origin")
[1] "2008-01-01 GMT"
attr(,"unit")
[1] "week"
```

Often we wish to use the minimum time realisation as the origin. The above call can be written more succinctly.

```
> seed = c ("2008-01-01", "2008-01-08", "2008-01-15", "2008-01-22")
> crtv (seed, relative=TRUE, unit="week")
[1] 0 1 2 3
attr(,"class")
[1] "crtv" "rtv"
attr(,"origin")
[1] "2008-01-01 GMT"
attr(,"unit")
[1] "week"
```

Assuming that we can create a drtv object from our seed object then we can compute number of years using the following:

$$\mathrm{f}_{\mathrm{year}}(x_i) - \mathrm{f}_{\mathrm{year}}(origin)$$

$$\mathrm{f}_{\mathrm{year}}(\bullet) = year + \frac{doy + \mathrm{f}_{\mathrm{day}}(\bullet) - 1}{\mathrm{n}_{\mathrm{year}}(year)}$$

As well as number of months using the following:

$$\mathrm{f}_{\mathrm{month}}(x_i) - \mathrm{f}_{\mathrm{month}}(origin)$$

$$\mathrm{f}_{\mathrm{month}}(\bullet) = 12\,year + month + \frac{day + \mathrm{f}_{\mathrm{day}}(\bullet) - 1}{\mathrm{n}_{\mathrm{month}}(month)}$$

Where

$$f_{\text{day}}(\bullet) = \frac{hour}{24} + \frac{minute}{1440} + \frac{second}{86400}$$

and

- x_i is a single time realisation.
- $origin$ is the origin of the time realisation.
- n_{year} is the number of days in the given year.
- n_{month} is the number of days in the given month.
- $\bullet$ is shorthand for any object which can be mapped to the argument list {year, month, day, hour, minute, second, dow, doy}.

We can produce inverses for these functions however they are messy. The reader can refer to the explode functions in the drtv.r source file if interested.

Based on these formulae we can create a crtv object with year or month as the unit (noting the effect of leap year on the example below).

```
> seed = c ("2000-01-01", "2000-07-02", "2001-01-01", "2001-07-02")
> crtv (seed, unit="year")
[1] 0.00000 0.50000 1.00000 1.49863
attr(,"class")
[1] "crtv" "rtv"
attr(,"origin")
[1] "2000-01-01 GMT"
attr(,"unit")
[1] "year"
```

Any rtv object can be tested and coerced to other objects. See the helps files for is.rtv and as.rtv for full details. A basic example is as.numeric (which is implemented as as.double.rtv). This strips a crtv object of its attributes leaving a numeric type.

```
> seed = c ("2000-01-01", "2000-07-02", "2001-01-01", "2001-07-02")
> x = crtv (seed, unit="year")
> as.numeric (x)
[1] 0.00000 0.50000 1.00000 1.49863
```

We may be interested in the opposite operation. Creating a crtv object from a numeric vector. This is accomplished using the default crtv constructor.

```
> v = c (0, 0.5, 1, 1.49863)
> x = crtv (v, unit="year")
> explicit.format (x)
[1] "2000-01-01" "2000-07-02" "2001-01-01" "2001-07-01"
```

Notice the error in the fourth realisation above. This is due to the round off error. Using greater precision we can avoid this, although errors are still likely when dealing with fractional seconds.

```
> x = crtv ("2001-07-02", unit="year")
> v = as.numeric (x)
> v
```

```
[1] 1.49863
> format (v, digits=16)
[1] "1.498630136986321"
> explicit.format (crtv (v, unit="year") )
[1] "2001-07-02"
```

As with formatting, the default origin and default unit can be changed by setting options. Such a call should be performed prior to any other rtv calls.

```
> #note that this code block is not evaluated
> #it has no effect on subsequent examples
> options (rtv.default.origin=as.POSIXct (crtv ("2009-01-01") ) )
> options (rtv.default.unit="second")
```

Now we are in a position to interconvert between discrete and continuous representations of time.

```
> seed = c ("2008-01-01", "2008-02-01", "2008-03-01", "2008-04-01")
> discrete.time = drtv (seed)
> discrete.time
$year
[1] 2008 2008 2008 2008

$month
[1] 1 2 3 4

$day
[1] 1 1 1 1

$hour
[1] 0 0 0 0

$minute
[1] 0 0 0 0

$second
[1] 0 0 0 0

$dow
[1] 2 5 6 2

$doy
[1]  1 32 61 92

attr(,"class")
[1] "drtv" "rtv"
> continuous.time = crtv (discrete.time)
> continuous.time
[1] 2922 2953 2982 3013
attr(,"class")
```

```
[1] "crtv" "rtv"
attr(,"origin")
[1] "2000-01-01 GMT"
attr(,"unit")
[1] "day"
> discrete.time = drtv (continuous.time)
> explicit.format (discrete.time)
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
```

We are also in a position to change origins or units.

```
> seed = c ("2008-01-01", "2008-01-02", "2008-01-03", "2008-01-04")
> x0 = crtv (seed, relative=TRUE)
> as.numeric (x0)
[1] 0 1 2 3
> x1 = crtv (x0, origin=attr (x0, "origin"), unit="hour")
> as.numeric (x1)
[1]  0 24 48 72
> x2 = crtv (x0, origin=crtv ("2007-12-31") )
> as.numeric (x2)
[1] 1 2 3 4
```

Notice the construct:

```
> origin=attr (x0, "origin")
```

Alternatively we could also use:

```
> origin=attr (x0, "unit")
```

In general these constructs are not necessary. However if we want to map one crtv object to another crtv object and copy either the origin or unit of the first crtv object (and the origin or unit are not a known constant in our script), we need something like this, otherwise the default origin and unit will be used.

One further command useful for working with crtv objects, is as.crtv. If the argument object is a crtv object, then it will be returned unchanged. Otherwise assuming a suitable seed object, a crtv object will be returned with the default origin and unit.

4 Mathematical Operations on rtv Objects

Most of the examples in this section are trivial. The important point to take note of, is that in general a function of an rtv object returns an rtv object. If the argument is a drtv object, then a drtv object is returned. If the argument is a crtv object, then a crtv object is returned. Note that some functions will convert drtv objects to crtv objects and then convert the result back to a drtv object. In these situations the default origin and default unit many effect the results.

Lets say we have the following drtv object:

```
> seed = paste ("2000-01-", 1:20, sep="")
> x = drtv (seed)
> explicit.format (x)
```

```
 [1] "2000-01-01" "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
 [6] "2000-01-06" "2000-01-07" "2000-01-08" "2000-01-09" "2000-01-10"
[11] "2000-01-11" "2000-01-12" "2000-01-13" "2000-01-14" "2000-01-15"
[16] "2000-01-16" "2000-01-17" "2000-01-18" "2000-01-19" "2000-01-20"
```

Perhaps the most common operations are combining and extracting. When combining rtv objects the return type will match the type of the first argument.

```
> explicit.format (c (x [1:5], crtv (0) ) )
[1] "2000-01-01" "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
[6] "2000-01-01"
```

We can also sample and sort.

```
> y = x [sample (1:20, 10)]
> explicit.format (y)
 [1] "2000-01-20" "2000-01-09" "2000-01-02" "2000-01-18" "2000-01-15"
 [6] "2000-01-11" "2000-01-16" "2000-01-19" "2000-01-06" "2000-01-08"
> y = sort (y)
> explicit.format (y)
 [1] "2000-01-02" "2000-01-06" "2000-01-08" "2000-01-09" "2000-01-11"
 [6] "2000-01-15" "2000-01-16" "2000-01-18" "2000-01-19" "2000-01-20"
```

Compute the mean.

```
> explicit.format (mean (x), date=FALSE)
[1] "2000-01-10 12:00:00"
```

Note the effects of formatting:

```
> explicit.format (mean (x) )
[1] "2000-01-10"
```

To avoid the problem associated with converting drtv objects to crtv and back, mentioned above, and also to avoid cluttered or misleading results due to formatting (e.g. if we hadn't set date=FALSE above), perform mathematical operations on crtv objects and don't format the output. However we can use as.numeric. Note that as.numeric is used here more than what one would reasonably use in practice. It serves to make the output more condensed and easier to read in a vignette setting.

So:

```
> #using default origin and unit
> x = crtv (x)


> as.numeric (mean (x) )
[1] 9.5
> as.numeric (range (x) )
[1]  0 19
> as.numeric (min (x) )
[1] 0
> as.numeric (max (x) )
```

```
[1] 19
```

Also note the effect of missing values.

```
> z = x
> z [10] = NA
> as.numeric (mean (z) )
[1] NA
> as.numeric (mean (z, na.rm=TRUE) )
[1] 9.526316
```

One exception to the rule of returning an rtv object is length (which returns the same value regards of whether the object is drtv or crtv). Another is diff. The range command also contains a diff argument, which by default is false. Another exception occurs when this is true.

```
> length (as.drtv (x) )
[1] 20
> length (as.crtv (x) )
[1] 20
> diff (x)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
> range (x, diff=TRUE)
[1] 19
```

We may also wish to add or subtract numeric values from rtv objects. Noting that adding an rtv object to another rtv object is not permitted.

The core function is rtv.incr, which allows us to choose units. If units are not specified then they default to day for drtv objects and the same unit for crtv objects.

```
> z = x [1:4]

> explicit.format (rtv.incr (z, 5) )
[1] "2000-01-06" "2000-01-07" "2000-01-08" "2000-01-09"
> explicit.format (rtv.incr (drtv (z), 5) )
[1] "2000-01-06" "2000-01-07" "2000-01-08" "2000-01-09"

> explicit.format (rtv.incr (z, 1, "year") )
[1] "2001-01-01" "2001-01-01" "2001-01-02" "2001-01-03"
> explicit.format (rtv.incr (crtv (z, unit="year"), 1) )
[1] "2001-01-01" "2001-01-01" "2001-01-02" "2001-01-03"
```

In general it is easier to work with expressions of the form $a + b$. Multiplication and division of rtv objects are also not permitted.

```
> explicit.format (z + 1)
[1] "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
> explicit.format (1 + z)
[1] "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05"
```

```
> explicit.format (z - 1)
[1] "1999-12-31" "2000-01-01" "2000-01-02" "2000-01-03"
```

Note that if one really wants to add two rtv objects, or multiply an rtv object, then an almost equivalent result can be obtained by casting them to numeric.

We can use the timeseq function if we wish to create sequences of (always crtv) time points. The first argument is an rtv object of length one or two. If the length is one, a second rtv object is required (of length one). The two values give the minimum and maximum values of the sequence. A third argument gives the number of points. Further arguments can also be used to specify the origin and unit of the resulting crtv object.

```
> as.numeric (timeseq (range (x), n=5) )
[1]  0.00  4.75  9.50 14.25 19.00
> as.numeric (timeseq (min (x), max (x), 5, unit="month") )
[1] 0.0000000 0.1532258 0.3064516 0.4596774 0.6129032
```

Equivalently, for the first sequence.

```
> as.numeric (min (x) + (0:4) * 19/4)
[1]  0.00  4.75  9.50 14.25 19.00
```

It is also possible to create sequences using other objects and coerce the result to an rtv object. However the above approaches are recommended.

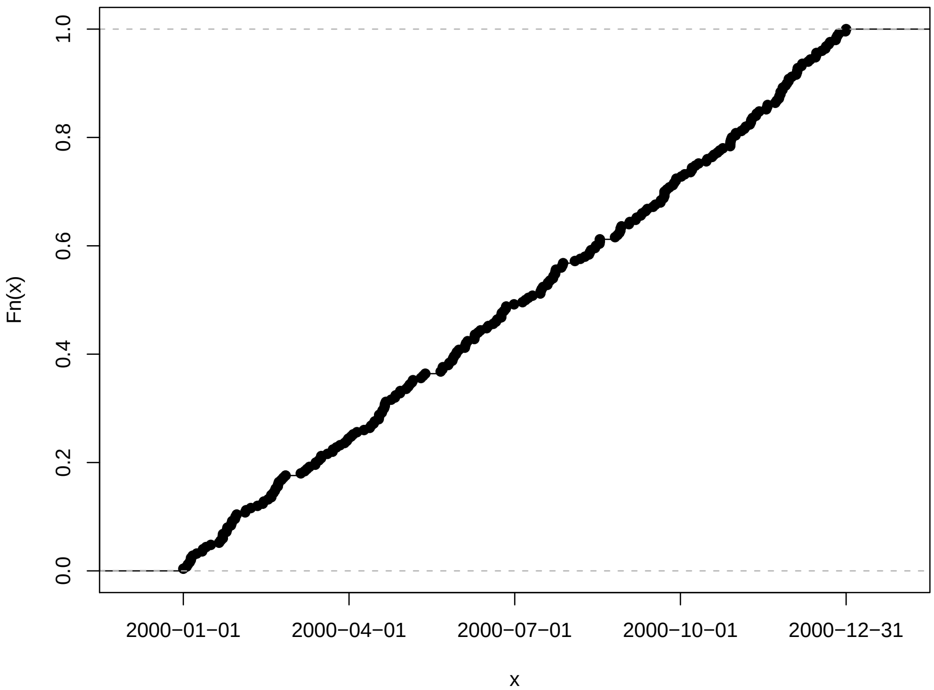
5 Simulation and Exploratory Data Analysis

Sometimes we may wish to simulate a time sample. The exact process for creating an rtv object with simulated realisations will depend on the distribution. Two examples are given. One for a uniform distribution and one for a normal distribution.

We can produce exploratory plots of the data. Such a plot can be enhanced using the timeaxis command (for plotting a timeaxis). The first argument is the side (refer to the help file for the axis function) and the second argument is the rtv object.

```
> #realisations from a uniform random time variable
> bound = crtv (c ("2000-01-01", "2001-01-01") )
> x = bound [1] + range (bound, diff=TRUE) * runif (250)
```

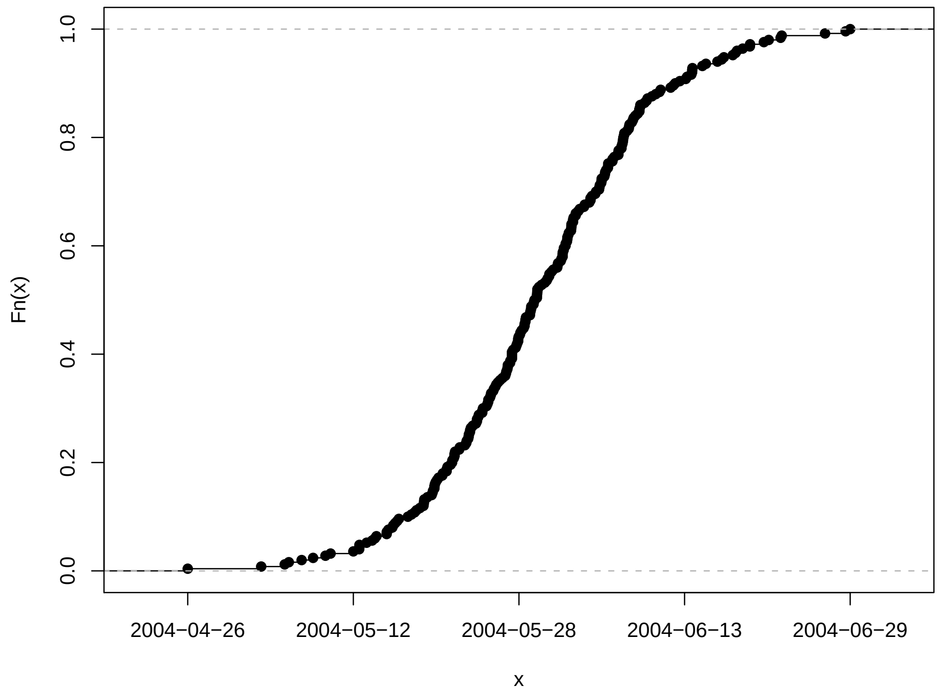
```
> plot (ecdf (x), main=NULL, axes=FALSE)
> box ()
> timeaxis (1, x)
> axis (2)
```



Simulated realisations from a uniform random time variable.

```
> #realisations from a normal random time variable
> #sd in days
> mu = crtv ("2004-06-01")
> sd = 10
> x = mu + rnorm (250, sd = sd)
```

```
> plot (ecdf (x), main=NULL, axes=FALSE)
> box ()
> timeaxis (1, x)
> axis (2)
```



Simulated realisations from a normal random time variable.

6 Calendar Operations

This following functions mainly exist as support functions for other functions given so far. However there are many situations when they may be useful in themselves.

Most a self explanatory, so commentary will be kept to a minimum. Note that all are vectorised and apply the recycling rule when arguments are of different lengths.

```
> year = 2000:2010
> is.leap (year)
 [1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
> ndays.year (year)
 [1] 366 365 365 365 366 365 365 365 366 365 365


> month = 1:12
> ndays.month (2000, month)
 [1] 31 29 31 30 31 30 31 31 30 31 30 31
> cumdays.month (2000, month)
 [1]  31  60  91 121 152 182 213 244 274 305 335 366


> date.to.dow (2000, 2, 1)
[1] 2
> date.to.doy (2000, 2, 1)
[1] 32
```

```
> doy.to.date (2000, 32)
$month
[1] 2

$day
[1] 1
```

We can also format the month or the day of the week using the functions month.string or dow.string. In both cases we can set the case by case="lower" or case="upper" (omitting or providing any other value results in title case). We can also set the number of letters by nletters = ...some value..., which by default is 3. Use NA for full names.

```
> month.string (month)
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> month.string (month, case="lower")
 [1] "jan" "feb" "mar" "apr" "may" "jun" "jul" "aug" "sep" "oct" "nov" "dec"


> dow = 1:7
> dow.string (dow)
[1] "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"
> dow.string (dow, case="upper", nletters=1)
[1] "M" "T" "W" "T" "F" "S" "S"
> dow.string (dow, nletters=NA)
[1] "Monday"    "Tuesday"   "Wednesday" "Thursday"  "Friday"    "Saturday"
[7] "Sunday"
```