package: rtv 0.4.0

Random Time Variables

Charlotte Maia

May 26, 2010

This vignette, introduces the rtv package, an R package for conveniently representing, manipulating and visualising time data. Here, time is regarded as a random variable, and rtv objects are used to represent realisations of that random variable. This is particularly useful for modelling change points, irregular timeseries and failure events. Here, rtv objects (which are defined by an abstract class and two subclasses) are similar to R's POSIXt objects. However rtv objects use/allow: (1) strong constructor overloading; (2) continuous representations of time, with user-specified origins and units (i.e. the number of days, weeks or years, since some reference event, possibly the first event in series of events); and (3) weaker formatting.

Introduction

The rtv package is designed for conveniently representing, manipulating and visualising time data. Perhaps the representation of time data is the most important, as it influences our ability interpret the data itself, as well as our ability to manipulate and visualise data. There are different ways that we can represent time data, and it makes sense to represent our data in a way that's convenient, given the nature and purpose of our data. For modelling purposes, most time variables (and time values) can be classified into three overlapping sets:

- **Timestamps and Formatted Text**

  Whilst there are different ways of representing timestamps, they're generally presented to the user as formatted text (e.g. "2004-04-18 10:05:27"). In general, timestamps contain both the date and the time of day, hence they tell us exactly when something happened.

- **Timeseries**

  In principle, a timeseries represents a series of time values, where each time value maps to some response value. Whilst many timeseries models partly ignore time values by regarding them as natural numbers, denoting the ith response or error value, time values are important for interpreting models in practice. They are also important for modelling irregular timeseries.

- **Waiting Times**

  With waiting times, often we are interested in expected waiting times, expected incidence rates, or perhaps the probability that a certain event will happen within a certain time period.

Reiterating, these sets overlap. An industrial dataset, may contain timestamps (as formatted text) and it may be of interest to use that dataset to compute change points or failure rates. This requires us to transform formatted text values into numerical values. Similar transformations may be required for timeseries modelling (especially for irregular timeseries). Plus it may be necessary to transform numerical values back to formatted text values so that we can interpret our model.

Given that the representation of time data is very important, that there are these different sets of time variables, and the need to transform back and forth between textual and numeric variables, the

rtv package seeks to represent time variables in a way that that generalises all these time variables and simplifies the transformation process.

In order to make this generalisation, time variables are generalised at two levels. Firstly, we use objects (in the computer science sense) to represent time values, something which is discussed in the next section. Secondly, we use random variables to model time.

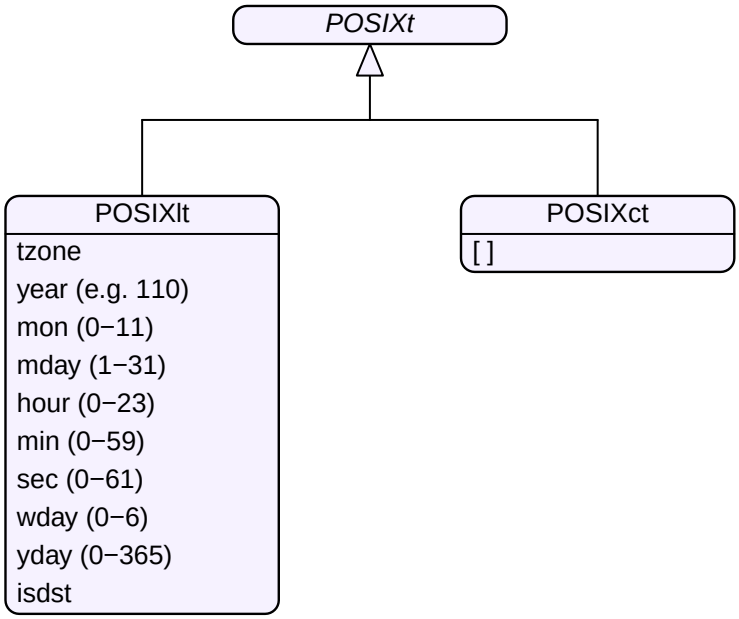
By regarding time as a random variable, we have a mathematical model that allows us to model all the time variables presented here. However, it's not sufficient to regard time as just any random variable. There are special issues associated with time, in particular time has relatively complex units, plus a tendency for cycles to appear in functions of time.

A random time variable, requires a random variable whose sample space and whose realisations, are special time numbers, that address those special issues. This is difficult to achieve using enumerations or real numbers, however it's clearly attainable if we use objects, which is part of justification for using objects.

In addition to their generality, there are further benefits to modelling time as random variables. Many important processes involve events that happen at random points in time. One of the major goals of this package, is to support modelling such processes.

Modelling Time as Objects

As mentioned earlier we need objects to represent time. This isn't a new idea (especially not in R). Many readers will be familiar with R's POSIXt objects. These are defined by the following classes (noting the square brackets in POSIXct attribute list imply itself):



These are powerful classes (and conform to several ISO standards), however, in the authors opinion they have the following problems:

1. Both the POSIXlt and POSIXct classes have very obscure constructors. A POSIXlt object can be created by with as.POSIXlt or strptime. A POSIXct object can be created with as.POSIXct or ISOdatetime.

2. In the POSIXlt class, firstly "year" represents the number of years since since 1900, so this year (2010) is year 110, secondly some (not all) of the enumerations start at zero (rather than one).

This is counter-intuitive in business and most other application areas. e.g. Month zero is January and month eleven is December.

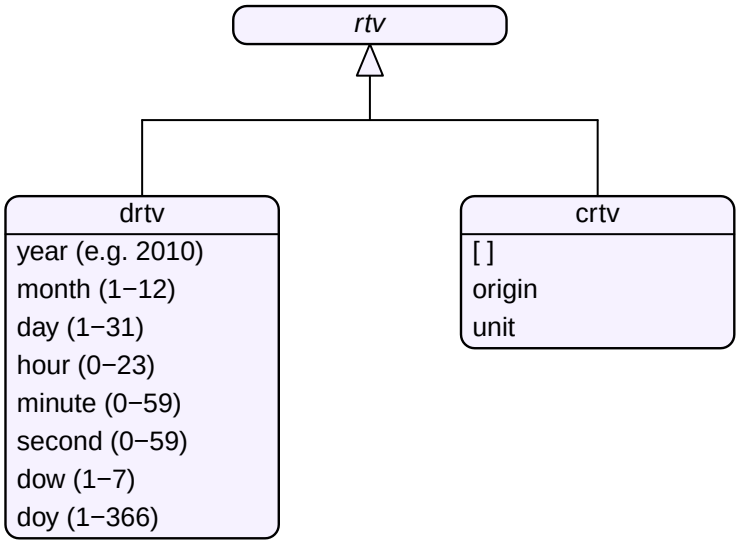
3. In the POSIXct class, time is represented by the number of seconds since "1970-01-01 00:00:00". Whilst by default, output is formatted, this representation is also counter-intuitive, plus it's inconvenient for modelling waiting times.

4. There's a function called difftime that computes the difference between two time values, where the units (e.g. days or weeks) can be specified by the user. However, the function doesn't allow months or years.

5. All POSIXt objects are sensitive to timezones in one way or another, noting that for most modelling purposes, we don't need timezones, plus timezones can cause unexpected results.

6. Printing POSIXt objects tends to produce highly formatted text, which obscures the representation of time.

In order to address these problems, plus better support modelling time as a random variable, the rtv package implements rtv objects. These are similar to the POSIXt objects (noting in some cases they even make use of them). Here, we have the following classes:



There are three classes, an abstract rtv class, a drtv class that roughly speaking extends rtv and list, and a crtv class that roughly speaking extends rtv and numeric. The diagram only shows the rtv classes. Noting that there are some intermediate classes that add functionality to the list and numeric classes, which we won't discuss.

In the drtv class, the "d" roughly implies discrete. The drtv class regards time as discrete and mostly-recursive units (i.e. a year is divided into months, which is divided into days, which is divided into hours, etc). This reflects the common everyday use of time. In the crtv class, the "c" roughly implies continuous. The crtv class regards time as a continuous (and potentially fractional) count of some time unit (e.g. days or years) that have happened since some reference event.

In response to the problems identified above, rtv objects:

1. Have obvious constructors, we call drtv or crtv to create an object. Noting the classes make strong use of constructor overloading.

2. In the drtv class, discrete time units have been designed to reflect their common use. Noting that dow is the day of the week, where 1 maps to Monday and 7 maps to Sunday.

3. In the crtv class, we have great flexibility, which we will discuss further.

4. In principle, there's no timezones. This ensures that the identity $x + 1$ day $= x + 24$ hours is always true.

5. Global options control formatting. By default, formatting is minimal.

Constructors for crtv objects, allow us to provide an origin and unit attribute. The origin is an arbitrary time value (as a drtv object, or anything that can be used to create a drtv object), and defaults to the equivalent of "2000-01-01 06:00:00". The unit is one of {year, month, day, hour, minute, second or week}, where day is the default. Using the default origin and unit, a value of 10 represents "2000-01-11 06:00:00", and can be interpreted as 10 days since the origin. Special consideration, has been given to using months and years as units. They are designed to accurately reflect the heterogeneous nature of the units.

A number of global options are mentioned in appendix A, plus the formulation of months and years in appendix B.

Whilst rtv objects are designed to represent both date and time of day values, we can also use them to represent date only values. For date only values, there's still information on the time of day, however we can partly ignore it.

We can't completely ignore it, because it can influence truncation and rounding operations (which are also subject to floating point errors). Noting that truncation is relatively common (i.e. formatting a time value to show the date only). The default behaviour of rtv constructors, when dealing with dates, is to set the time of day, to 6am.

Creating drtv Objects

We can create drtv objects, from a variety of seed objects. Following is a list of drtv constructors (don't take too much notice of the word "methods"), along with the function signature for drtv.character (which we are going to focus on):

```
> methods (drtv)
[1] drtv.character drtv.crtv      drtv.Date      drtv.default   drtv.drtv
[6] drtv.POSIXct   drtv.POSIXlt
> args (drtv.character)
function (x, ..., date = TRUE, hour = 6, style)
NULL
```

We can create a drtv object (from a text seed aka R character vector), using the default arguments as follows:

```
> seed = c ("2008-01-01", "2008-02-01", "2008-03-01", "2008-04-01")
> x = drtv (seed)
> x
  year month day hour minute second dow doy
1 2008     1   1    6      0      0   2   1
2 2008     2   1    6      0      0   5  32
3 2008     3   1    6      0      0   6  61
4 2008     4   1    6      0      0   2  92
```

By default, printing a drtv (or crtv) object, doesn't use a lot of formatting. This is intentional, and is intended to expose the structure of the object, and (hopefully) make them simple and intuitive to use. We can produce a formatted version, using drtvf. Noting drtvf, stands for create a drtv object and format it.

```
> drtvf (x)
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
```

As mentioned earlier a drtv object extends both rtv and list. We can access the object's attributes like a list, however in many ways they have been designed to feel like vectors:

```
> x$day
[1] 1 1 1 1
> length (x)
[1] 4
> x [1]
  year month day hour minute second dow doy
1 2008     1   1    6      0      0   2   1
```

To create a drtv (date-based) object, using midnight rather than 6am, we can use the function drtvs (which simply calls the same constructor, using the argument hour=0). Noting that the "s" stands for simple.

```
> drtvs (seed)
  year month day hour minute second dow doy
1 2008     1   1    0      0      0   2   1
2 2008     2   1    0      0      0   5  32
3 2008     3   1    0      0      0   6  61
4 2008     4   1    0      0      0   2  92
```

We can also create a drtv object, using both dates and times of the day, using the drtvx function (which also calls the same constructor, however using the argument date=FALSE). Noting that the "x" stands for extended.

```
> seedx = c ("2010-06-08 14:45:00", "2010-06-08 04:22:00",
         "2010-06-08 22:45:00", "2010-06-08 02:05:00")
> x = drtvx (seedx)
> x
  year month day hour minute second dow doy
1 2010     6   8   14     45      0   2 159
2 2010     6   8    4     22      0   2 159
3 2010     6   8   22     45      0   2 159
4 2010     6   8    2      5      0   2 159
```

To produce a formatted representation of the object, including all the information, we need an extra argument for drtvf:

```
> drtvf (x, date=FALSE)
[1] "2010-06-08 14:45:00" "2010-06-08 04:22:00" "2010-06-08 22:45:00"
[4] "2010-06-08 02:05:00"
```

If this seems like too much work, we can change global options.

```
> rtvo.format (TRUE)
> rtvo.date (FALSE)
> x
[1] "2010-06-08 14:45:00" "2010-06-08 04:22:00" "2010-06-08 22:45:00"
[4] "2010-06-08 02:05:00"
```

```
> rtvo.reset ()
```

Furthermore, we can specify a style argument, this is the same as the the format argument used by strptime. If style is provided, both the date and hour arguments are ignored.

Creating crtv Objects

We create crtv objects, in a similar way to drtv objects. We also have a variety of constructors, including a constructor for text seeds. We have crtvs and crtvx functions similar to the drtvs and drtvx functions, plus still use drtvf to force formatting.

```
> methods (crtv)
[1] crtv.character crtv.crtv      crtv.Date      crtv.default   crtv.drtv
[6] crtv.POSIXct   crtv.POSIXlt
> args (drtv.character)
function (x, ..., date = TRUE, hour = 6, style)
NULL


> x = crtv (seed)
> drtvf (x)
[1] "2008-01-01" "2008-02-01" "2008-03-01" "2008-04-01"
> x
[1] 2922 2953 2982 3013
(origin="2000-01-01", unit="day")
```

Furthermore, all the crtv constructors, allow us to specify a seed (first argument), an origin (second argument) and a unit (third argument). Note that the origin, can be any value that is a valid drtv seed. Also note, that where a text seed is used (as in the following example) it will assume a date only value, using a 6am default hour. As a general rule, when creating crtv objects using crtvx, the origin is best specified using either drtvs or drtvx.

```
> #date only
> crtv (seed, "2008-01-01", "hour")
[1]    0  744 1440 2184
(origin="2008-01-01", unit="hour")


> #date and time of day - origin at midnight
> crtvx (seedx, drtvs ("2008-01-01"), "hour")
[1] 21350.75 21340.37 21358.75 21338.08
(origin="2008-01-01", unit="hour")


> #date and time of day - origin = first recorded event
> crtvx (seedx, drtvx (seedx [1]), "hour")
[1]   0.00000 -10.38333   8.00000 -12.66667
(origin="2010-06-08", unit="hour")


> #date and time of day - origin = first occurring event
> crtvx (seedx, min (crtvx (seedx) ), "hour")
```

```
[1] 12.666667  2.283333 20.666667  0.000000
(origin="2010-06-08", unit="hour")
```

Sometimes, we may wish to create a crtv object representing a sequence of values (not random as such, however still useful):

```
> rng = range (x)
> seq (rng [1], rng [2], 10)
 [1] 2922.000 2932.111 2942.222 2952.333 2962.444 2972.556 2982.667 2992.778
 [9] 3002.889 3013.000
(origin="2000-01-01", unit="day")
```

Note that we can extract the origin and unit attributes.

```
> x$origin
  year month day hour minute second dow doy
1 2000     1   1    6      0      0   6   1
> x$unit
[1] "day"
```

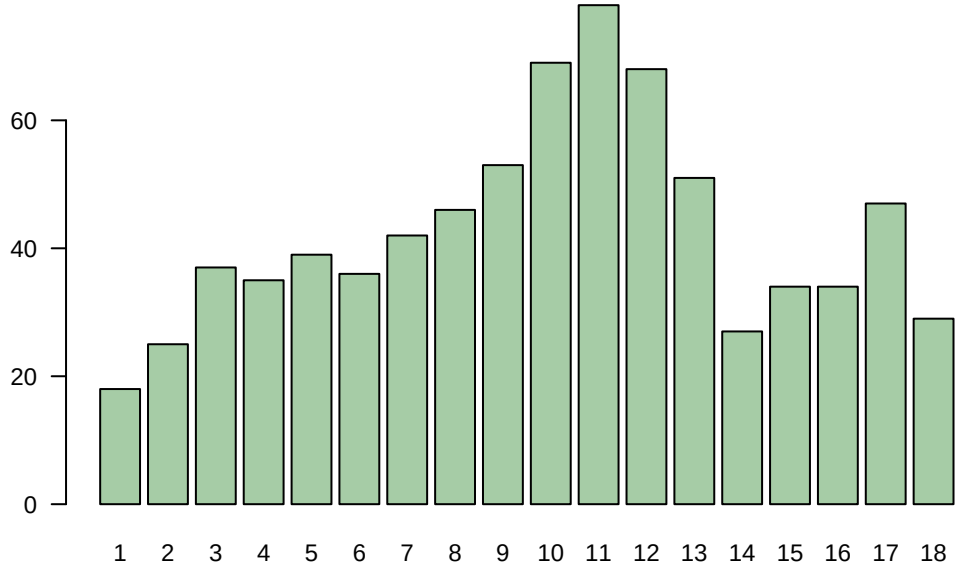
Univariate Distributions

Lets say we have a random time variable with realisations $\mathbf{x}$, corresponding to random events from the beginning of 2007 to the middle of 2008.

```
> #first 5 and last 5
> n = length (x)
> drtvf (sort (x) [c (1:5, (n-4):n)], FALSE)
 [1] "2007-01-01 09:03:45" "2007-01-01 12:21:57" "2007-01-03 09:46:30"
 [4] "2007-01-06 23:40:01" "2007-01-13 06:15:27" "2008-06-25 15:02:42"
 [7] "2008-06-25 22:27:38" "2008-06-27 07:16:36" "2008-06-28 22:05:33"
[10] "2008-06-29 00:59:23"
```

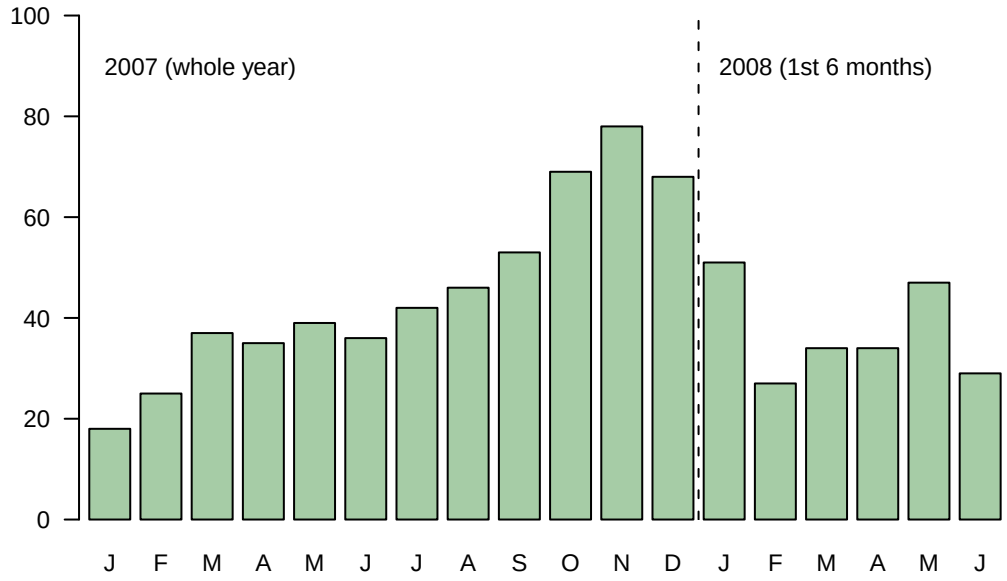
Let's say that we are interested in the variable's distribution, with respect to monthly units. Perhaps the most intuitive way to visualise the distribution, and perhaps the easiest to interpret, might be a barplot.

```
> u = floor (x) + 1
> k = table (u)
> barplot (k, names=names (k), col=rgb (0.65, 0.8, 0.65) )
```



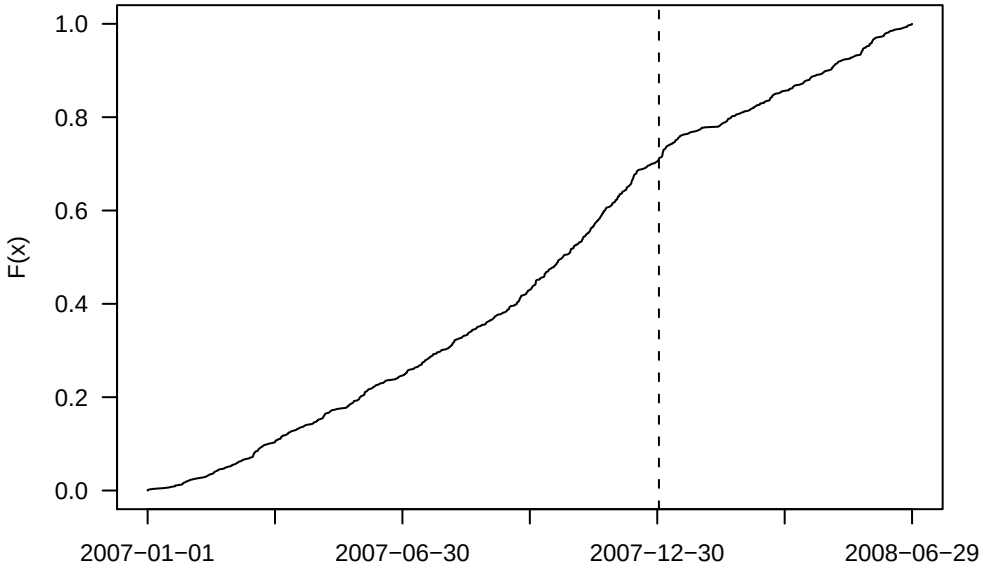
That's kind of ugly, perhaps a more beautiful version...

```
> labs = formatmonth (c (1:12, 1:6), nchars=1)
> barplot (k, names=labs, xaxs="i", col=rgb (0.65, 0.8, 0.65),
         xlim=c (0, 18), ylim= c (0, 100), width=0.8, space=0.25)
> abline (v=12.1, lty=2)
> text (c (0.5, 12.5), c (90, 90),
         c ("2007 (whole year)", "2008 (1st 6 months)"), adj=c (0, 0.5) )
```



Alternatively, we can use the default plot, which produces a plot of an ecdf (with a formatted time axis). Noting that convex segments imply increasing frequencies, and concave segments imply decreasing frequencies.

```
> plot (x, ylab="F(x)")
> abline (v=12, lty=2)
```



Irregular Timeseries

Lets say we have another random time variable with realisations **x** and some other random response variable with realisations **y**. Lets say that the **x** values span five days, and the **y** values, well doesn't matter. Note that the third argument in the plot function, is the positions of the tick marks.

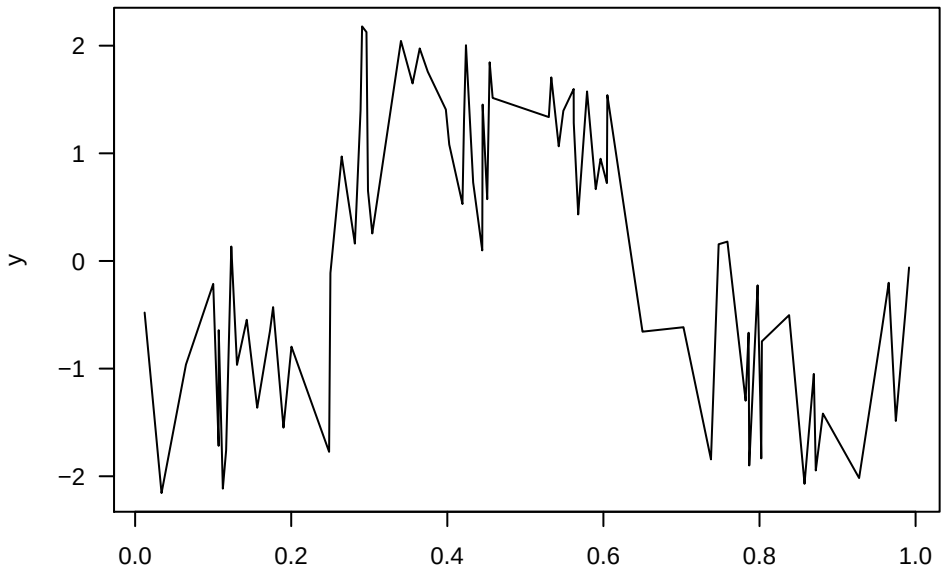
```
> #first 5 and last 5
> n = length (x)
> drtvf (sort (x) [c (1:5, (n-4):n)], FALSE)
 [1] "2010-04-05 03:45:13" "2010-04-05 05:57:54" "2010-04-05 06:21:06"
 [4] "2010-04-05 07:17:32" "2010-04-05 08:31:52" "2010-04-09 14:31:17"
 [7] "2010-04-09 15:36:07" "2010-04-09 20:06:38" "2010-04-09 22:15:48"
[10] "2010-04-09 22:57:31"

> plot (x, y, at=(0:4) + 0.5)
> abline (v=1:4, col="grey20", lty=3)
```

Cyclic Timeseries

Here cycle refers to any repeating process, and specifically in the context of functions of rtv objects, a function that repeats (or tends to repeat) itself over one of the crtv units. Using the previous example, we can examine a daily cycle:

```
> plot (x, y, cycle="day")
```



Manipulating rtv Objects

Some examples (many of which are only defined for crtv objects):

```
> seed = c ("2008-01-01", "2008-01-04", "2008-01-03", "2008-01-02")
> x = crtv (seed, seed [1], "hour")
> x
[1]  0 72 48 24
(origin="2008-01-01", unit="hour")


> length (x)
[1] 4
> formatdow (drtv (x [1])$dow)
[1] "Tue"
> c (x, x)
[1]  0 72 48 24  0 72 48 24
(origin="2008-01-01", unit="hour")
> rep (x, 4)
 [1]  0 72 48 24  0 72 48 24  0 72 48 24  0 72 48 24
(origin="2008-01-01", unit="hour")
> sort (x)
```

```
[1]  0 24 48 72
(origin="2008-01-01", unit="hour")
> order (x)
[1] 1 4 3 2
> mean (x)
[1] 36
(origin="2008-01-01", unit="hour")
> min (x)
[1] 0
(origin="2008-01-01", unit="hour")
> max (x)
[1] 72
(origin="2008-01-01", unit="hour")
> range (x)
[1]  0 72
(origin="2008-01-01", unit="hour")
> round (x)
[1]  0 72 48 24
(origin="2008-01-01", unit="hour")
> x [2]
[1] 72
(origin="2008-01-01", unit="hour")
> x + 24
[1] 24 96 72 48
(origin="2008-01-01", unit="hour")
```

Calendar Operations

The following functions mainly exist as support functions, both for manipulating time values, and for formatting them. They're reasonably self-explanatory. Note that they are vectorised.

```
> year = 2001:2010
> is.leap (year)
 [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
> year [is.leap (year)]
[1] 2004 2008
> ndays.year (year)
 [1] 365 365 365 366 365 365 365 366 365 365

> month = 1:12
> ndays.month (2007, month)
 [1] 31 28 31 30 31 30 31 31 30 31 30 31
> ndays.month (2008, month)
```

```
 [1] 31 29 31 30 31 30 31 31 30 31 30 31
```

```
> date2dow (2010, 2, 1:10)
 [1] 1 2 3 4 5 6 7 1 2 3
> date2doy (2010, 2, 1:10)
 [1] 32 33 34 35 36 37 38 39 40 41
> doy2date (2010, 24:35)
$month
 [1] 1 1 1 1 1 1 1 1 2 2 2 2

$day
 [1] 24 25 26 27 28 29 30 31  1  2  3  4
```

```
> dow = 1:7
> formatdow (dow)
[1] "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"
> formatdow (dow, case="upper", nchars=NA)
[1] "MONDAY"    "TUESDAY"   "WEDNESDAY" "THURSDAY"  "FRIDAY"    "SATURDAY"
[7] "SUNDAY"
```

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

Appendix A:
Global Options

The rtv package uses several global options. They control the way that the rtv constructors interpret textual representations of time, they way that time is printed and formatted, and the default origin and unit. Following is a list of the options, along with their class, default value and description.

Option	Cl	Default	Description
rtv.origin	D	"2000-01-01 06:00:00"	Default origin.
rtv.unit	C	"day"	Default unit.
rtv.format	L	FALSE	When printing rtv objects, call drtvf, to produce highly formatted text.
rtv.date	L	TRUE	Should the rtv format method, include the date only.
rtv.styled	C	"%Y-%m-%d"	For formatted text, that includes the date only, the style of the text.
rtv.stylex	C	"%Y-%m-%d %H:%M:%OS"	For formatted text, that includes both the date and the time of day, the style of text.

Where, Cl refers to class, D refers to drtv (noting the default value shown, is the formatted version), L refers to logical, and C refers to character. In addition to the options, are the following utility functions, to help set the options:

```
> rtvo.reset ()
> rtvo.format (format=FALSE)
> rtvo.date (date=FALSE)
```

The first, resets all the options to their defaults, the others set the corresponding options.

Appendix B:
Monthly and Yearly Values

Given a crtv value in days, deriving the equivalent value in weeks or seconds is trivial. However, deriving the equivalent value in months or years requires careful consideration. Here, rather than assume that a month or a year, corresponds to some constant and approximate number of days, we regard one month as the amount of time required from the kth day of the xth month, to the same kth day of the $(x+1)$th month. e.g. Jan 1st and Feb 1st (of the same year), are exactly one month apart. The same principle applies to years.

Using fresh notation, assuming that we can create a drtv object x for some time values and another drtv object k for some origin, then the number of months d_m and the number of years d_y, can be computed as follows:

$$\begin{aligned} \mathrm{d}_y(x|k) &= \mathrm{f}_y(x) - \mathrm{f}_y(k) \\ \mathrm{d}_m(x|k) &= \mathrm{f}_m(x) - \mathrm{f}_m(k) \end{aligned}$$

Where:

$$\begin{aligned} \mathrm{f}_y(\bullet) &= year + (doy + \mathrm{f}_d(\bullet) - 1)/n_{year} \\ \mathrm{f}_m(\bullet) &= (12)(year) + month + (day + \mathrm{f}_d(\bullet) - 1)/n_{month} \\ \mathrm{f}_d(\bullet) &= hour/24 + minute/1440 + second/86400 \end{aligned}$$

and where,

- Is a drtv object.

n_{year} Is the number of days in the given years.

n_{month} Is the number of days in the given months.

Noting that $\bullet$ has been used as shorthand notation. Where $\bullet$ has been used as a function's argument to represent a drtv object, the object's attributes {year, month, day, hour, minute, second, dow, doy}, have also been regarded as arguments.

We can produce inverses for these functions however they're messy. The reader can refer the source files if interested.