Motivation

Illustration: Time series computations for NASDAQ-100 series.

Infrastructure: “zoo” series with “Date” time index for daily data.

Technical details: Deferred to later sections.


Data source: The data are taken from http://finance.yahoo.com/. In fact, this can be queried interactively using `get.hist.quote()` from `tseries`.

> Data in external text file `ndx.txt` where each row of the file looks like this:
> 03 Jan 1995| 398.00
> 04 Jan 1995| 399.65
> ...

> Read into “zoo” time series:
> 
> R> ndx <- read.zoo("ndx.dat", format = "%d %b %Y", sep = "|")
> R> plot(ndx)
Motivation

Compute log-difference returns (in percent):

\[
R > \text{rndx} \leftarrow 100 * \text{diff}(\text{log(ndx)})
\]

Visualize:

\[
R > \text{plot(cbind(ndx, rndx))}
\]

Different visualization: \texttt{xyplot()} from \texttt{lattice}.

\[
R > \text{library("lattice")}
R > \text{xyplot(cbind(ndx, rndx))}
\]

Or: \texttt{autoplot()} from \texttt{ggplot2}.

\[
R > \text{library("ggplot2")}
R > \text{autoplot(cbind(ndx, rndx)) + facet_free()}
\]

Rolling mean (adjusted to annual level):

\[
R > \text{plot(rollapply(rndx, 365, mean) * 365.25)}
R > \text{abline(h = 0, lty = 2)}
\]
Motivation

Extract subset:

```r
R> ndx2 <- window(ndx,
+ start = as.Date("2000-01-01"), end = as.Date("2000-01-15"))
R> ndx2
  3790.55 3546.20 3507.31 3340.81 3529.60
  3717.41 3544.35 3478.14 3612.08 3704.74
```

Extract time index and data:

```r
R> time(ndx2)
R> coredata(ndx2)
[1] 3790.55 3546.20 3507.31 3340.81 3529.60 3717.41 3544.35
[8] 3478.14 3612.08 3704.74
```

Union:

```r
R> merge(ndx2, lag(ndx2, k = -1))
ndx2 lag(ndx2, k = -1)
2000-01-03 3790.55 NA
2000-01-04 3546.20 3790.55
2000-01-05 3507.31 3546.20
2000-01-06 3340.81 3507.31
2000-01-07 3529.60 3340.81
2000-01-10 3717.41 3529.60
2000-01-11 3544.35 3717.41
2000-01-12 3478.14 3544.35
2000-01-13 3612.08 3478.14
2000-01-14 3704.74 3612.08
```
Motivation

Intersection:

\[ \text{R> } \text{merge(ndx2, lag(ndx2, k = -1), all = FALSE)} \]

\begin{verbatim}
ndx2  lag(ndx2, k = -1)
2000-01-04  3546.20     3790.55
2000-01-05  3507.31     3546.20
2000-01-06  3340.81     3507.31
2000-01-07  3529.60     3340.81
2000-01-10  3717.41     3529.60
2000-01-11  3544.35     3717.41
2000-01-12  3478.14     3544.35
2000-01-13  3612.08     3478.14
2000-01-14  3704.74     3612.08
\end{verbatim}

Extend along daily grid (via empty "zoo" series):

\[ \text{R> td <- seq(from = start(ndx2), to = end(ndx2), by = "day")} \]
\[ \text{R> ndx2 <- merge(zoo(, td), ndx2)} \]
\begin{verbatim}
  3790.55  3546.20  3507.31  3340.81  3529.60
    NA     3592.20  3529.60  3478.14  3612.08
2000-01-13 2000-01-14
    3612.08     3704.74
\end{verbatim}

Various strategies for dealing with NAs: linear interpolation, last observation carried forward.

\[ \text{R> merge(ndx2, na.approx(ndx2), na.locf(ndx2))} \]
\begin{verbatim}
ndx2  na.approx(ndx2)  na.locf(ndx2)
2000-01-03  3790.55      3790.55     3790.55
2000-01-04  3546.20      3546.20     3546.20
2000-01-05  3507.31      3507.31     3507.31
2000-01-06  3340.81      3340.81     3340.81
2000-01-07  3529.60      3529.60     3529.60
2000-01-08  NA           3592.20     3529.60
2000-01-09  NA           3654.81     3529.60
2000-01-10  3717.41      3717.41     3717.41
2000-01-11  3544.35      3544.35     3544.35
2000-01-12  3478.14      3478.14     3478.14
2000-01-13  3612.08      3612.08     3612.08
2000-01-14  3704.74      3704.74     3704.74
\end{verbatim}

Furthermore: \textbf{na.omit}().

Compress to weekly data:

\[ \text{R> to_monday <- function(x)
  + 7 * ceiling(as.numeric(x - 3)/7) + as.Date(-3)} \]
\[ \text{R> ndx_d <- aggregate(ndx, to_monday, tail, 1)} \]
\begin{verbatim}
  401.59      410.48      410.13      407.22      416.14
  430.77      425.66      428.42      436.68      441.76
\end{verbatim}

This exploits that the origin (1970-01-01, \texttt{as.Date(0)}) is a Thursday.

Compress to monthly data:

\[ \text{R> ndx_m <- aggregate(ndx, as.yearmon, mean)} \]
\[ \text{R> head(ndx_m, 10)} \]
\begin{verbatim}
  408.007    425.107    444.035    453.099    484.352    517.553    567.712
  575.515    594.547    579.728
\end{verbatim}
In finance, business and economics, time-annotated data is ubiquitous. Therefore, a fundamental building block for more complex structures in finance/business/economics are times and dates.

In (macro-)economics, the most important types of times are/were years, quarters and months.

In finance, required times are more often at daily or intra-day level.

**Times and dates: Data structures**

- **Classes**: Implementations of time/date objects that capture all necessary information of underlying conceptual entities.
- **Methods**: Computations and extraction of relevant information.
- **Note**: In principle, use of documented methods is preferred but sometimes knowledge about the internal structure is very useful.

**Typical actions and operations:**

- set up time from numeric or character input,
- extract underlying numeric scale,
- produce character label,
- use for plotting,
- sequences of times with given spacing,
- time differences,
- move forward/backward on time scale,
- comparison (less/greater).
Times and dates: Data structures

Typical actions and operations (and R functions/generics):
- set up time from numeric or character input: class constructors,
- extract underlying numeric scale: `as.numeric()` method,
- produce character label: `format()` and `as.character()` method,
- use for plotting: input for `plot()` method, e.g., `Axis()` method,
- sequences of times with given spacing: `seq()` method,
- time differences: group generic functions or `difftime()`,
- move forward/backward on time scale: group generic functions,
- comparison (less/greater): group generic functions.

Time: Years

**Example:** 1997, 1998, 2002, 2004, ...

**Class:** “numeric” or (even better) “integer”

```r
R> ty
R> as.character(ty)
```

```r
[1] 1
R> ty + 1
```

Time: Quarters

**Example:** 2000 Q1, 2001 Q3, 2002 Q2, 2002 Q3, ...

**Class:** “numeric” (first attempt)

```r
R> as.character(tq)
[1] "2000 Q1" "2001 Q3" "2002 Q2" "2002 Q3"
```

**Class:** “yearqtr” (improved)

```r
R> tq <- as.yearqtr(tq)
R> as.character(tq)
[1] "2000 Q1" "2001 Q3" "2002 Q2" "2002 Q3"
```

```r
[1] 1.5
R> tq + 1/4
[1] "2000 Q2" "2001 Q4" "2002 Q3" "2002 Q4"
```
Time: Quarters

Idea: “numeric” vector with class attribute that handles matching/rounding correctly and provides coercion to many other classes.

Class constructor:

```r
yearqtr <- function(x)
  structure(floor(4 * x + .001)/4, class = "yearqtr")
```

provided in package zoo.

Time: Months


Class: “yearmon” (analogous to “yearqtr”)

```r
R> tm <- yearmon(c(2000, 2001, 2001, 2002) + c(0, 9, 11, 7)/12)
R> tm
```

```r
R> as.yearmon(2000)
[1] "Jan 2000"
```

```r
R> as.yearmon("2000 Jan", format = "%Y %b")
[1] "Jan 2000"
```

Time: Days

Example: 1970-01-01, 2001-07-12, 2005-03-24, …

Class: “Date” (number of days since 1970-01-01)

```r
R> td <- as.Date(c("1970-01-01", "2001-07-12", "2005-03-24"))
R> td
```

```r
R> as.numeric(td)
[1] 0 11515 12866
```

```r
R> as.character(td)
```

Time: Days

```r
R> format(as.Date(td), "%B %d, %Y")
```

```r
Time difference of 11515 days
R> td + 1
```

```r
R> as.Date(2)
[1] "1970-01-03"
```
Time: Intra-day

Example: 1970-01-01 00:00:00, 2001-07-12 12:23:59, ...

Class: "chron" (days since 1970-01-01, without time zone or daylight savings time)

R> tc <- chron(c(0, 11515 + 12/24 + 23/1440 + 59/86400))
R> tc
[1] (01/01/70 00:00:00) (07/12/01 12:23:59)
R> as.character(tc)
[1] "(01/01/70 00:00:00)" "(07/12/01 12:23:59)"
R> as.numeric(tc)
[1] 0.0 11515.5
R> paste(format(as.Date(dates(tc))), format(times(tc)%%1))
[1] "1970-01-01 00:00:00" "2001-07-12 12:23:59"
Time in days:
[1] 11515.5

Times and dates: Coercion

Coercion: conversions between different time/date formats.

R> as.Date(tq)
R> as.yearqtr(td)
[1] "1970 Q1" "2001 Q3" "2005 Q1"
R> as.yearmon(tp)
R> as.Date(tp)
[1] "1970-01-01" "2001-07-12"
R> as.chron(tp)
[1] (01/01/70 00:00:00) (07/12/01 12:23:59)

Note: Some care is required when time zones and daylight savings times are involved.

R> as.POSIXct(tq)
[1] "2000-01-01 01:00:00 CET" "2001-07-01 02:00:00 CEST"
[3] "2002-04-01 02:00:00 CEST" "2002-07-01 02:00:00 CEST"
R> as.POSIXct(td)
[1] "1970-01-01 01:00:00 CET" "2001-07-12 02:00:00 CEST"
[3] "2005-03-24 01:00:00 CET"
R> as.POSIXct(tc)
[1] "1970-01-01 01:00:00 CET" "2001-07-12 14:23:58 CEST"

Time: Intra-day

Example: 1970-01-01 00:00:00 GMT, 2001-07-12 12:23:59 GMT, ...

Class: "POSIXct" (seconds since 1970-01-01, with time zone and daylight savings time)

R> tp <- as.POSIXct(c(0, 994940639), + origin = "1970-01-01", tz = "GMT")
R> tp
[1] "1970-01-01 00:00:00 GMT" "2001-07-12 12:23:59 GMT"
R> as.numeric(tp)
[1] 0 994940639
R> as.character(tp)
[1] "1970-01-01 00:00:00" "2001-07-12 12:23:59"
R> format(tp, `%B %d, %Y (%H:%M:%S)`)
[1] "January 01, 1970 (00:00:00)" "July 12, 2001 (12:23:59)"
Time difference of 11515.5 days
**Times and dates: Coercion**

**Simpler:** When time zones and daylight savings time are not required, all computations can be made simpler by setting the local time zone `TZ` to GMT.

```r
R> Sys.setenv(TZ = "GMT")
R> as.POSIXct(tq)
R> as.POSIXct(td)
R> as.POSIXct(tc)
[1] "1970-01-01 00:00:00 GMT" "2001-07-12 12:23:58 GMT"
```

**Time: Intra-day**

**Class:** “timeDate” (built on “POSIXct”, internal computations done in GMT, deals with time zones and daylight savings times via financial centers)

```r
R> timeDate(tp)
GMT
[1] [1970-01-01 00:00:00] [2001-07-12 12:23:59]
R> timeDate(tp, FinCenter = "Zurich")
Zurich
[1] [1970-01-01 01:00:00] [2001-07-12 14:23:59]
R> timeDate(tp, FinCenter = "NewYork")
NewYork
[1] [1969-12-31 19:00:00] [2001-07-12 08:23:59]
```

**Times and dates: Summary**

**Classes:**

- “numeric”/“integer” (base): annual/quarterly/monthly.
- “yearqtr”/“yearmon” (zoo): quarterly/monthly.
- “Date” (base): daily.
- “chron” (chron): intra-day. No time zones, daylight savings time. Slightly non-standard interface.
- “POSIXct” (base): intra-day. With time zones, daylight savings time. Computations in GMT are straightforward. Other time zones might require some more attention.
- “timeDate” (timeDate): intra-day. With time zones, daylight savings time via concept of financial centers.

**Recommendation:** Use time/date class that is appropriate for your data (and not more complex). See Grothendieck and Petzoldt (2004).
Time series: Structure

Time/date objects are usually not interesting as standalone objects but are used to annotate other data.

The most important application of this are time series where there is for each time point a vector of (typically numeric) observations.

The observations are most easily arranged in a vector (of length \( n \)) or an \( n \times k \) matrix whose elements are ordered and indexed by \( n \) different times/dates.

Types of time series:
- **irregular** (unequally spaced),
- **strictly regular** (equally spaced),
- or have an underlying regularity, i.e., be created from a regular series by omitting some observations.

For strictly regular series: the whole time index can be reconstructed from start, end and time difference between two observations. The reciprocal value of the time difference is also called frequency.

For irregular series: all time indexes need to be stored in a vector of length \( n \).

Time series: Implementations

There are many implementations for time series data in R.

Virtually all of them are focused on numeric data and fix some particular class for the time index. The most important are:
- **"ts"** (base): regular “numeric” time index (e.g., annual, quarterly, monthly),
- **"its"** (its): irregular time index of class “POSIXct”,
- **"irts"** (tseries): irregular time index of class “POSIXct”,
- **"timeSeries"** (timeSeries): irregular time index of class “timeDate”,
- **"zoo"** (zoo): regular or irregular time index of arbitrary class.
- **"xts"** (xts): built on top of “zoo”, with specialized infrastructure for time indexes of class “Date”, “POSIXct”, “chron”, “timeDate”, “yearmon”, “yearqtr”, ...

Advantages of “zoo”:
- Can be used with arbitrary time indexes (i.e., you could also provide your own specialized class).
- Standard interface: employs R’s standard methods and introduces only few new generics.
- Talks to all other classes: coercion functions are available, e.g., `as.ts()`, `as.timeSeries()`, etc. that work if the time index is of the required class. The reverse `as.zoo()` always works.

Recommendations:
- Use “zoo” for storage and basic computations.
- If necessary, coerce to other classes for analysis etc.
- “xts” is helpful extension of “zoo” for date/time indexes.
Time series: Operations

Typical operations for time series:
- visualization,
- extraction of observations or associated times,
- lags and differences,
- subsets in a certain time window,
- union and intersection of several time series,
- aggregation along a coarser time grid,
- rolling computations such as means or standard deviations.

Time series: Implementations

How do arbitrary time indexes in “zoo” work?
To provide the desired functionality, few actions are required:
- ordering,
- matching,
- combining,
- subsetting,
- querying length n.

Typical operations for time series (and suitable R generics):
- visualization: plot(),
- extraction of observations or associated times:
  time() (and coredata()),
- lags and differences: lag() and diff(),
- subsets in a certain time window: window(),
- union and intersection of several time series: merge(),
- aggregation along a coarser time grid: aggregate(),
- rolling computations such as means or standard deviations:
  (rollapply()).

How do arbitrary time indexes in “zoo” work?
To provide the desired functionality, few actions are required:
- ordering: ORDER() (by default calling the non-generic order()),
- matching: MATCH() (by default calling the non-generic match()),
- combining: c(),
- subsetting: [,]
- querying length n: length().
Time series: Implementations

If suitable methods are available for the chosen time index (including all time/date classes above), all tasks (merging, aggregating, etc.) can be performed without any knowledge about the particular time index class.

For some special operations, further methods are useful/necessary, e.g., `axis()`/`Axis()`, `as.numeric()`, `as.character()`.

If the time index is of a particular class, coercions to and from other time series classes can be easily provided.

References


