**zoo Quick Reference**

**Abstract**

This vignette gives a brief overview of (some of) the functionality contained in *zoo* including several nifty code snippets when dealing with (daily) financial data. For a more complete overview of the package’s functionality and extensibility see Zeileis and Grothendieck (2005) (contained as vignette “zoo” in the package), the manual pages and the reference card.

**Keywords**: irregular time series, daily data, weekly data, returns.

---

**Read a series from a text file**

To read in data in a text file, `read.table()` and associated functions can be used as usual with `zoo()` being called subsequently. The convenience function `read.zoo` is a simple wrapper to these functions that assumes the index is in the first column of the file and the remaining columns are data.

Data in `demo1.txt`, where each row looks like

23 Feb 2005|43.72

can be read in via

```
R> inrusd <- read.zoo("demo1.txt", sep = ",", format = "%d %b %Y")
```

The `format` argument causes the first column to be transformed to an index of class "Date".

The data in `demo2.txt` look like

```
Daily,24 Feb 2005,2055.30,4337.00
```

and requires more attention because of the format of the first column.

```
R> tmp <- read.table("demo2.txt", sep = ",")
R> z <- zoo(tmp[, 3:4], as.Date(as.character(tmp[, 2]), format = "%d %b %Y"))
R> colnames(z) <- c("Nifty", "Junior")
```

**Query dates**

To return all dates corresponding to a series `index(z)` or equivalently

```
R> time(z)
```

```
```

can be used. The first and last date can be obtained by
R> start(z)
[1] "2005-02-10"

R> end(inrusd)
[1] "2005-03-10"

**Convert back into a plain matrix**
To strip off the dates and just return a plain vector/matrix *coredata* can be used

R> plain <- coredata(z)
R> str(plain)

```
num [1:20, 1:2] 2063 2082 2098 2090 2062 ...
- attr(*, "dimnames")=List of 2
 ..$ : chr [1:20] "1" "2" "3" "4" ...
 ..$ : chr [1:2] "Nifty" "Junior"
```

**Union and intersection**
Unions and intersections of series can be computed by *merge*. The intersection are those days where both series have time points:

R> m <- merge(inrusd, z, all = FALSE)

whereas the union uses all dates and fills the gaps where one series has a time point but the other does not with NAs (by default):

R> m <- merge(inrusd, z)

cbind(inrusd, z) is almost equivalent to the *merge* call, but may lead to inferior naming in some situations hence *merge* is preferred

To combine a series with its lag, use

R> merge(inrusd, lag(inrusd, -1))

```
inrusd lag(inrusd, -1)
2005-02-10  43.78   NA
2005-02-11  43.79  43.78
2005-02-14  43.72  43.79
2005-02-15  43.76  43.72
2005-02-16  43.82  43.76
2005-02-17  43.74  43.82
2005-02-18  43.84  43.74
2005-02-21  43.82  43.84
2005-02-22  43.72  43.82
2005-02-23  43.72  43.72
2005-02-24  43.70  43.72
2005-02-25  43.69  43.70
2005-02-28  43.64  43.69
2005-03-01  43.72  43.64
```
Visualization

By default, the `plot` method generates a graph for each series in `m`

```r
R> plot(m)
```

but several series can also be plotted in a single window.

```r
R> plot(m[,2:3], plot.type = "single", col = c("red", "blue"), +     lwd = 2)
```
Select (a few) observations

Selections can be made for a range of dates of interest

```r
R> window(z, start = as.Date("2005-02-15"), end = as.Date("2005-02-28"))
```

Nifty Junior
2005-02-15 2089.95 4367.25
2005-02-17 2061.90 4320.15
2005-02-18 2055.55 4318.15
2005-02-21 2043.20 4282.25
2005-02-22 2058.40 4326.10
2005-02-23 2057.10 4346.00
2005-02-24 2055.30 4337.00
2005-02-25 2060.90 4305.75
2005-02-28 2103.25 4388.20

and also just for a single date

```r
R> m[as.Date("2005-03-10")]
```

inrusd Nifty Junior
2005-03-10 43.58 2167.4 4648.05

Handle missing data

Various methods for dealing with NAs are available, including linear interpolation

```r
R> interpolated <- na.approx(m)
```

‘last observation carried forward’,

```r
R> m <- na.locf(m)
R> m
```
Prices and returns

To compute log-difference returns in %, the following convenience function is defined

\[
R > \text{prices2returns} \leftarrow \text{function(x) 100 * diff(log(x))}
\]

which can be used to convert all columns (of prices) into returns.

\[
R > r \leftarrow \text{prices2returns}(m)
\]

A 10-day rolling window standard deviations (for all columns) can be computed by

\[
R > \text{rollapply}(r, 10, \text{sd})
\]

<table>
<thead>
<tr>
<th></th>
<th>inrusd</th>
<th>Nifty</th>
<th>Junior</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005-02-10</td>
<td>0.14599121</td>
<td>0.6993355</td>
<td>0.7878843</td>
</tr>
<tr>
<td>2005-02-11</td>
<td>0.14527421</td>
<td>0.6300543</td>
<td>0.8083622</td>
</tr>
<tr>
<td>2005-02-12</td>
<td>0.14115862</td>
<td>0.8949318</td>
<td>1.0412806</td>
</tr>
<tr>
<td>2005-02-13</td>
<td>0.15166883</td>
<td>0.9345299</td>
<td>1.0256508</td>
</tr>
<tr>
<td>2005-02-14</td>
<td>0.14285470</td>
<td>0.9454103</td>
<td>1.1957959</td>
</tr>
<tr>
<td>2005-02-15</td>
<td>0.13607992</td>
<td>0.9453855</td>
<td>1.1210963</td>
</tr>
<tr>
<td>2005-02-16</td>
<td>0.11962991</td>
<td>0.9334899</td>
<td>1.1105966</td>
</tr>
<tr>
<td>2005-02-17</td>
<td>0.11963193</td>
<td>0.8585071</td>
<td>0.9388661</td>
</tr>
<tr>
<td>2005-02-18</td>
<td>0.09716262</td>
<td>0.8569891</td>
<td>0.9131822</td>
</tr>
<tr>
<td>2005-02-19</td>
<td>0.09787943</td>
<td>0.8860388</td>
<td>1.0566389</td>
</tr>
<tr>
<td>2005-02-20</td>
<td>0.11568119</td>
<td>0.8659890</td>
<td>1.0176645</td>
</tr>
</tbody>
</table>

To go from a daily series to the series of just the last-traded-day of each month \text{aggregate} can be used.
R> prices2returns(aggregate(m, as.yearmon, tail, 1))

inrusd Nifty Junior
Mar 2005 -0.1375831 3.004453 5.752866

Analogously, the series can be aggregated to the last-traded-day of each week employing a convenience function `nextfri` that computes for each "Date" the next friday.

R> nextfri <- function(x) 7 * ceiling(as.numeric(x - 1)/7) + as.Date(1)
R> prices2returns(aggregate(na.locf(m), nextfri, tail, 1))

<table>
<thead>
<tr>
<th>Date</th>
<th>Inrusd</th>
<th>Nifty Junior</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005-02-18</td>
<td>0.11411618</td>
<td>-1.2809533</td>
</tr>
<tr>
<td>2005-02-25</td>
<td>-0.34273997</td>
<td>0.2599329</td>
</tr>
<tr>
<td>2005-03-04</td>
<td>0.04576659</td>
<td>4.1464226</td>
</tr>
<tr>
<td>2005-03-11</td>
<td>-0.29785794</td>
<td>0.8921286</td>
</tr>
</tbody>
</table>

**Query Yahoo! Finance**

When connected to the internet, Yahoo! Finance can be easily queried using the `get.hist.quote` function in

R> library("tseries")

From version 0.9-30 on, `get.hist.quote` by default returns "zoo" series with a "Date" attribute (in previous versions these had to be transformed from "ts" 'by hand').

A daily series can be obtained by:

R> sunw <- get.hist.quote(instrument = "SUNW", start = "2004-01-01", + end = "2004-12-31")

A monthly series can be obtained and transformed by

R> sunw2 <- get.hist.quote(instrument = "SUNW", start = "2004-01-01", + end = "2004-12-31", compression = "m", quote = "Close")

Here, "yearmon" dates might be even more useful:

R> time(sunw2) <- as.yearmon(time(sunw2))

The same series can equivalently be computed from the daily series via

R> sunw3 <- aggregate(sunw[, "Close"], as.yearmon, tail, 1)

The corresponding returns can be computed via

R> r <- prices2returns(sunw3)

where `r` is still a "zoo" series.

**Query Oanda**


A daily series of EUR/USD exchange rates can be queried by
R> eur.usd <- get.hist.quote(instrument = "EUR/USD", provider = "oanda", 
+       start = "2004-01-01", end = "2004-12-31")

This contains the exchange rates for every day in 2004. However, it is common practice in many situations to exclude the observations from weekends. To do so, we write a little convenience function which can determine for a vector of "Date" observations whether it is a weekend or not

R> is.weekend <- function(x) ((as.numeric(x) - 2)%%7) < 2

Based on this we can omit all observations from weekends

R> eur.usd <- eur.usd[!is.weekend(time(eur.usd))]

The function is.weekend introduced above exploits the fact that a "Date" is essentially the number of days since 1970-01-01, a Thursday. A more intelligible function which yields identical results could be based on the "POSIXlt" class

R> is.weekend <- function(x) {
+   x <- as.POSIXlt(x)
+   x$wday > 5 | x$wday < 1
+ }

References