Time Index

A time series is a series of data points indexed in time order. In R, all data types for which an order is defined can be used to index a time series. If the operator < is defined for a data type, then the data type can be used to index a time series.

Date

```
today <- Sys.Date()     # current Date
yesterday <- today - 1  # subtract 1 day
yesterday < today       # the order is defined for Date
```

```
## [1] TRUE
```

POSIXct

```
now <- Sys.time()       # current time
ago <- now - 3600       # subtract 3600 seconds
ago <  now              # the order is defined for POSIXct
```

```
## [1] TRUE
```

Character

```
'a' < 'b'               # the order is defined for character
```

```
## [1] TRUE
```

Numeric

```
1 < 2                   # the order is defined for numeric
```

```
## [1] TRUE
```

Complex

```
2+0i < 1+3i             # the order is NOT defined for complex
```

```
## Error in 2 + (0+0i) < 1 + (0+3i): invalid comparison with complex values
```

The ‘zoo’ Package

The `zoo` package consists of the methods for totally ordered indexed observations. All indexes discussed above can be used. The package aims at performing calculations containing irregular time series of numeric vectors, matrices and factors. The package is an infrastructure that tries to do all basic things well, but it doesn’t provide modeling functionality.

```
# install the package
install.packages('zoo')
```

```
# load the package
require(zoo)
```

The below set of exercises shows some of zoo concepts.

Declaration

```
# create a unidimensional zoo object indexed by default
zoo(x = c(100,123,43,343,22))
```

```
##   1   2   3   4   5
```

```
## 100 123  43 343  22

# create a unidimensional zoo object indexed by numeric
x <- c(100, 123, 43, 343, 22)
i <- c(0, 0.2, 0.4, 0.5, 1)
zoo(x = x, order.by = i)

##   0 0.2 0.4 0.5   1
## 100 123  43 343  22

# create a unidimensional zoo object indexed by character
x <- c(100, 123, 43, 343, 22)
i <- c('z', 'b', 'd', 'c', 'a')
zoo(x = x, order.by = i)

##   a   b   c   d   z
##  22 123 343  43 100

# create a multidimensional zoo object indexed by Date
x <- data.frame('price' = c(100,99.3,100.2), 'volume' = c(9.9,1.3,3.6))
i <- as.Date(c('2018/01/01', '2018/02/23', '2018/05/01'), format = "%Y/%m/%d")
zoo(x = x, order.by = i)

##            price volume
## 2018-01-01 100.0    9.9
## 2018-02-23  99.3    1.3
## 2018-05-01 100.2    3.6

# create a multidimensional zoo object indexed by POSIXct
x <- data.frame('price' = c(100,99.3,100.2), 'volume' = c(9.9,1.3,3.6))
i <- as.POSIXct(c('20180101 120631', '20180223 085145', '20180501 182309'),
format = "%Y%m%d %H%M%S")
zoo(x = x, order.by = i)

##                     price volume
## 2018-01-01 12:06:31 100.0    9.9
## 2018-02-23 08:51:45  99.3    1.3
## 2018-05-01 18:23:09 100.2    3.6
```

Manipulation

```
# assign colnames
x <- data.frame(c(100,99.3,100.2), c(9.9,1.3,3.6))
z <- zoo(x = x)
colnames(z) <- c('p','v')
z

##       p   v
## 1 100.0 9.9
## 2  99.3 1.3
## 3 100.2 3.6

# assign indexes
index(z) <- as.Date(c('2018/01/01', '2018/02/23', '2018/05/01'), format =
"%Y/%m/%d")
z

##                p   v
## 2018-01-01 100.0 9.9
## 2018-02-23  99.3 1.3
```

```
## 2018-05-01 100.2 3.6

# starting index
start(z)

## [1] "2018-01-01"

# ending index
end(z)

## [1] "2018-05-01"

# select specific indexes
i <- as.Date(c('2018-01-01', '2018-05-01'))
z[i]

##                p   v
## 2018-01-01 100.0 9.9
## 2018-05-01 100.2 3.6

# select specific columns
z$p                # equivalent to z[,'p']

## 2018-01-01 2018-02-23 2018-05-01
##      100.0       99.3      100.2

# change the 2nd observation 'p' value
z$p[2] <- 105      # equivalent to z[2,'p'] <- 105
z

##                p   v
## 2018-01-01 100.0 9.9
## 2018-02-23 105.0 1.3
## 2018-05-01 100.2 3.6

# subset the series
window(z, start = '2018-01-01', end = '2018-03-1')

##              p   v
## 2018-01-01 100 9.9
## 2018-02-23 105 1.3

# increments
diff(z)

##               p    v
## 2018-02-23  5.0 -8.6
## 2018-05-01 -4.8  2.3

# lag the series
lag(z, k = 1)      # shift the time base back

##                p   v
## 2018-01-01 105.0 1.3
## 2018-02-23 100.2 3.6

# lag the series
lag(z, k = -1)     # shift the time base forward

##              p   v
## 2018-02-23 100 9.9
```

```
## 2018-05-01 105 1.3
```

```
# merge series
z.next   <- lag(z, k =  1)
z.prev   <- lag(z, k = -1)
z.merged <- merge(z, z.next, z.prev)
z.merged
```

```
##              p.z v.z p.z.next v.z.next p.z.prev v.z.prev
## 2018-01-01 100.0 9.9    105.0      1.3       NA       NA
## 2018-02-23 105.0 1.3    100.2      3.6      100      9.9
## 2018-05-01 100.2 3.6       NA       NA      105      1.3
```

```
# handle missing data. Approx with previous non-NA value
na.locf(z.merged)
```

```
##              p.z v.z p.z.next v.z.next p.z.prev v.z.prev
## 2018-01-01 100.0 9.9    105.0      1.3       NA       NA
## 2018-02-23 105.0 1.3    100.2      3.6      100      9.9
## 2018-05-01 100.2 3.6    100.2      3.6      105      1.3
```

```
# handle missing data. Approx with next non-NA value
na.locf(z.merged, fromLast = TRUE)
```

```
##              p.z v.z p.z.next v.z.next p.z.prev v.z.prev
## 2018-01-01 100.0 9.9    105.0      1.3      100      9.9
## 2018-02-23 105.0 1.3    100.2      3.6      100      9.9
## 2018-05-01 100.2 3.6       NA       NA      105      1.3
```

```
# handle missing data. Drop NA
z.merged[complete.cases(z.merged),]
```

```
##            p.z v.z p.z.next v.z.next p.z.prev v.z.prev
## 2018-02-23 105 1.3    100.2      3.6      100      9.9
```

Arithmetic operations are performed element-by-element on matching indexes of the two zoo obejcts. If the operation involves a zoo and a vector object, then the operation is performed on the whole zoo object.

```
x <- matrix(101:112, nrow = 3, ncol = 4, byrow = TRUE)
z <- zoo(x)
# add 1 to the whole series
z + 1
```

```
##
## 1 102 103 104 105
## 2 106 107 108 109
## 3 110 111 112 113
```

```
# multiply the first observation by 0, the second one by 1 and the third one by
2
z * c(0,1,2)
```

```
##
## 1   0   0   0   0
## 2 105 106 107 108
## 3 218 220 222 224
```

```
# compute the increments
z - lag(z, -1)  # equivalent to diff(z)
```

```
##
## 2 4 4 4 4
## 3 4 4 4 4
```

```
# compute the percentage increments
z / lag(z, -1) - 1
```

```
##
## 2 0.03960396 0.03921569 0.03883495 0.03846154
## 3 0.03809524 0.03773585 0.03738318 0.03703704
```

```
# compute the rolling mean on a 2-observation window
rollapply(z, width = 2, FUN = mean)
```

```
##
## 1 103 104 105 106
## 2 107 108 109 110
```

The 'xts' Package

The `xts` package provides an extensible time series class, enabling uniform handling of many R time series classes by extending `zoo`. An `xts` object can be indexed by the `Date`, `POSIXct`, `chron`, `yearmon`, `yearqtr`, `DateTime` data types but not by `numeric` or `character`.

```
# install the package
install.packages('xts')
```

```
# load the package
require(xts)
```

The methods seen for `zoo` objects can be applied to `xts`. The below set of exercises shows some of additional xts specific concepts.

```
# create an xts object
dates <- seq(as.Date("2017-05-01"), length=1000, by="day")          # generate a
sequence of dates
data  <- c(price = cumprod(1+rnorm(1000, mean = 0.001, sd = 0.01))) # generate
some random data
x <- xts(x = data, order.by = dates)                                # create the
xts object
colnames(x) <- 'price'                                               # assign
colnames
head(x)                                                              # print the
first observations
```

```
##                price
## 2017-05-01 0.9953952
## 2017-05-02 0.9940995
## 2017-05-03 1.0105887
## 2017-05-04 1.0123118
## 2017-05-05 1.0146329
## 2017-05-06 1.0330492
```

```
# change format of time display
indexFormat(x) <- "%Y/%m/%d"
head(x)
```

```
##                price
## 2017/05/01 0.9953952
```

```
## 2017/05/02 0.9940995
## 2017/05/03 1.0105887
## 2017/05/04 1.0123118
## 2017/05/05 1.0146329
## 2017/05/06 1.0330492

# estimate frequency of observations
periodicity(x)

## Daily periodicity from 2017-05-01 to 2020-01-25

# first observation
first(x)

##                price
## 2017/05/01 0.9953952

# last observation
last(x)

##               price
## 2020/01/25 3.039245

# first 3 days of the last week of data
first(last(x, '1 week'), '3 days')

##               price
## 2020/01/20 3.059311
## 2020/01/21 3.059618
## 2020/01/22 3.095431

# convert to OHLC
# valid periods are "seconds", "minutes", "hours", "days", "weeks", "months",
"quarters","years"
x.ohlc <- to.period(x, period = 'quarters')
head(x.ohlc)

##               x.Open   x.High     x.Low  x.Close
## 2017/06/30 0.9953952 1.106982 0.9940995 1.106982
## 2017/09/30 1.1025282 1.217479 1.0792620 1.137135
## 2017/12/31 1.1268060 1.268922 1.1245544 1.233901
## 2018/03/31 1.2586082 1.574023 1.2307955 1.574023
## 2018/06/30 1.5432948 1.632296 1.5026029 1.574151
## 2018/09/30 1.6134651 1.940108 1.5884681 1.865520

# calculate the yearly mean
ep <- endpoints(x.ohlc, on = "years")
period.apply(x.ohlc , INDEX = ep, FUN = mean)

##              x.Open   x.High    x.Low  x.Close
## 2017/12/31 1.074910 1.197794 1.065972 1.159339
## 2018/12/31 1.565839 1.813814 1.542075 1.747158
## 2019/12/31 2.227582 2.608671 2.177939 2.480005
## 2020/01/25 2.954804 3.095431 2.932865 3.039245
```
