# Introduction to R's time series facilities

Michael Lundholm*

September 22, 2011
Version 1.3

## 1 Introduction

This is an introduction to R's time series facilities. In an elementary way we deal with reading time series data into R, how time series objects are defined and their properties and finally how time series data can be manipulated. This is *not* an introduction to time series analysis. The note is written as a complement to my colleague Mahmood Arai's "A Brief Guide to R for Beginners in Econometrics" available at `http://people.su.se/~ma/R_intro/R_Brief_Guide.pdf`. It is presumed that R is installed and that the reader has some basic R–knowledge. An interactive R–session run parallel to reading the notes is recommended.

## 2 Preliminaries

Time series data have the property of being temporally ordered. Each individual observation have a date and these dates are organised sequentially. We consider here only the case when the time interval between any two observations with sequentially following dates is constant (i.e., the same between all observations). This means that we have a time index $t \in \{1, 2, \dots, T-1, T\}$, where 1 is the first date and $T$ is the date of the last observation (i.e., the length of the time series). The number of observations per year in a time series is called its *frequency*.

The basic function in R that defines time series is `ts()`. It has as its default frequency `frequency=1`. The starting date is by default year one; `start=1`. The following simple example creates a time series with frequency 1 (yearly data) of the first 10 positive integers, where we have written out the default values:

```
> ts(1:10, start = 1, frequency = 1)
```

*Department of Economics, Stockholm University, michael.lundholm@ne.su.se.

```
Time Series:
Start = 1
End = 10
Frequency = 1
 [1]  1  2  3  4  5  6  7  8  9 10
```

Naturally we can set the starting year arbitrarily:

```
> ts(1:10, start = 1998, frequency = 1)
```

```
Time Series:
Start = 1998
End = 2007
Frequency = 1
 [1]  1  2  3  4  5  6  7  8  9 10
```

In many instances the interval may be a fraction of a year, say a month or a quarter, so that we 12 or 4 observations per year. That is, the frequency is 12 or 4. Consider quarterly data:

```
> ts(1:10, start = 1998, frequency = 4)
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
1998    1    2    3    4
1999    5    6    7    8
2000    9   10
```

The main point with the function `ts()` is that it defines a time series object which consists of the data and a time line (including frequency). There is no need to define time as a specific variable or use specific time variables in existing data. We will below see how this property can be exploited.

## 3 Data

For illustrative purposes we will use the Swedish Consumer Price (CPI) with monthly index numbers from January 1980 to December 2005. This data can be downloaded from the Statistics Sweden web page `http://www.scb.se`. However, the data has to be generated from a data base and is therefore also available at `http://people.su.se/~lundh/reproduce/PR0101B1.scb`.[1] Start by downloading the files `PR0101B1.scb` to a working directory at your local computer:[2]

[1] Please note the curious naming convention that the data base of Statistics Sweden is using regarding file names; the base name in upper-case letters and numbers and the extension in lower case letters.

[2] For details about making data available to R see M. Lundholm, "Loading data into R", Version 1.1. August 18, 2010, `http://people.su.se/~lundh/reproduce/loading_data.pdf`.

```
> library(utils)
> URL <- "http://people.su.se/~lundh/reproduce/"
> FILE <- "PR0101B1.scb"
> download.file(paste(URL, FILE, sep = ""), FILE)
```

Then load the data file PR0101B1.scb in your favourite text editor and inspect it. The first 10 rows of the file look like this:

```
NA
1980M01       95.3
1980M02       96.8
1980M03       97.2
1980M04       97.9
1980M05       98.2
1980M06       98.5
1980M07       99.3
1980M08       99.9
1980M09       102.7
```

NA

```
> cpi <- read.table(FILE, skip = 1, col.names = c("Time",
+     "CPI"))
> head(cpi)
```

```
     Time  CPI
1 1980M01 95.3
2 1980M02 96.8
3 1980M03 97.2
4 1980M04 97.9
5 1980M05 98.2
6 1980M06 98.5
```

```
> str(cpi)
```

```
'data.frame':	312 obs. of  2 variables:
 $ Time: Factor w/ 312 levels "1980M01","1980M02",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ CPI : num  95.3 96.8 97.2 97.9 98.2 ...
```

where the data is assigned the name `cpi`. Note that the resulting object is a data frame.

The command `head(x)` gives the first 6 lines of the object `x` to which it is applied. The dates of the observations are in column 1 which has been assigned the name `Time` and the CPI is in column 2 with the name `CPI`.

# 4 Creating time series objects with `ts()`

The default of the function `ts()` is `ts(x,start=1,frequency=1)`, where `x` can be a vector, a matrix or a data frame. In order to create a time series object out the object `cpi` (where the first column is redundant) and were the starting date i January 1980 we use the following redefinition:

```
> (cpi1 <- ts(cpi[, 2], start = c(1980, 1), frequency = 12))

       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct
1980  95.3  96.8  97.2  97.9  98.2  98.5  99.3  99.9 102.7 104.2
1981 107.2 109.3 109.8 110.5 111.2 111.6 112.6 113.5 114.3 115.0
1982 117.4 119.0 119.3 120.1 120.7 121.1 121.9 122.2 122.9 124.6
1983 129.1 128.8 129.3 130.3 131.1 131.8 132.9 133.5 134.5 135.6
1984 139.4 138.9 140.9 141.8 142.8 142.4 142.8 143.9 144.8 145.5
1985 149.6 151.0 152.1 152.7 154.5 153.9 153.8 153.8 154.5 155.5
1986 158.9 159.0 158.7 159.7 159.7 159.7 160.1 159.9 161.3 161.9
1987 164.4 164.4 164.7 165.1 165.2 164.9 166.9 167.8 169.4 170.1
1988 171.6 172.9 173.7 175.2 175.8 176.3 177.1 177.5 178.8 180.2
1989 183.0 184.0 184.7 186.5 187.3 187.9 187.9 188.7 190.2 191.8
1990 199.0 199.9 205.4 205.2 206.4 206.2 208.2 209.6 212.0 213.4
1991 218.9 225.0 225.8 227.1 227.3 227.0 227.1 226.7 229.2 230.1
1992 230.2 230.3 231.3 231.9 232.0 231.5 231.2 231.3 234.6 235.1
1993 241.0 241.6 242.7 243.7 243.1 242.3 241.9 242.3 244.5 245.2
1994 245.1 245.9 246.8 247.8 248.3 248.4 248.4 248.5 250.7 251.0
1995 251.3 252.3 253.3 255.0 255.3 255.1 254.8 254.5 256.2 256.9
1996 255.6 255.8 257.0 257.6 257.3 256.3 255.7 254.5 256.0 255.9
1997 254.6 254.2 255.2 257.0 257.0 257.4 257.3 257.4 259.8 259.6
1998 256.9 256.6 257.0 257.7 258.1 257.6 257.0 255.7 256.8 257.3
1999 256.2 256.3 257.3 257.9 258.3 258.7 257.6 257.6 259.4 259.7
2000 257.5 258.7 259.9 260.0 261.3 261.2 260.0 260.2 262.0 262.6
2001 261.7 262.6 264.6 266.9 268.7 268.3 266.9 267.6 269.9 269.1
2002 268.8 269.4 271.8 272.9 273.6 273.2 272.3 272.4 274.5 275.4
2003 276.0 278.4 279.8 278.8 278.5 277.7 276.8 276.7 278.7 278.9
2004 278.0 277.3 279.4 279.4 280.1 278.9 278.5 278.2 280.2 281.0
2005 277.9 279.2 279.8 280.2 280.3 280.4 279.4 279.9 281.9 282.4
       Nov   Dec
1980 104.8 105.2
1981 115.4 114.9
1982 125.6 125.9
1983 136.4 137.5
1984 146.4 148.8
1985 156.5 157.1
1986 161.9 162.3
1987 170.7 170.7
```

```
1988 180.5 180.9
1989 192.2 192.8
1990 214.1 213.9
1991 231.1 230.8
1992 234.0 234.9
1993 245.3 244.3
1994 250.8 250.4
1995 256.8 256.0
1996 255.3 254.9
1997 259.2 259.1
1998 256.7 256.2
1999 259.0 259.6
2000 262.7 262.5
2001 269.2 269.5
2002 274.7 275.1
2003 278.3 278.6
2004 279.4 279.4
2005 281.7 281.8
```

```
> str(cpi1)
```

```
 Time-Series [1:312] from 1980 to 2006: 95.3 96.8 97.2 97.9 98.2 ...
```

Note first that `cpi[,2]` picks the entire second column. The argument `start=c(1980,1)` specifies the starting date as the date of the first observation in 1980. Since `frequency=12` defines data as monthly, the first observation in 1980 is January 1980. The resulting object is a time series object.

The `ts()` function has two clones: `is.ts()` which checks whether an object is a time series object and `as.ts()` which coerces an object to be a time series object:

```
> is.ts(cpi1)
```

```
[1] TRUE
```

```
> is.ts(cpi)
```

```
[1] FALSE
```

```
> is.ts(as.ts(cpi))
```

```
[1] TRUE
```

Note `cpi` is not a time series object to R even if it contains information about the dates of the observation. To R these dates are just factors.

Note finally that even if `cpi1` is a time series object, a part of this object selected through indexing loose these properties:

```
> cpi1[7:18]

 [1]  99.3  99.9 102.7 104.2 104.8 105.2 107.2 109.3 109.8 110.5
[11] 111.2 111.6

> is.ts(cpi1[7:18])

[1] FALSE
```

i.e., the latter being an attempt to access data from July 1980 to June 1981. Well, we do but the resulting object does not inherit the relevant time series properties. We will have to use other methods to consider subsets of time series objects.

## 5 Extracting attributes from time series objects

There are several commands that can extract different attributes from a time series object. Of course one can extract the time line (i.e., the dates of the observations):

```
> cpi1.time <- time(cpi1)
> head(cpi1.time)

[1] 1980.0 1980.1 1980.2 1980.2 1980.3 1980.4
```

We see that the time index is given as decimal numbers. The difference between two points in time (time difference) is a fixed number which depends on the frequency (i.e., it is the inverse of the frequency). The time difference between two observations in a time series object and the frequency of the object can of course be extracted:

```
> deltat(cpi1)

[1] 0.083333

> frequency(cpi1)

[1] 12
```

and we verify that $\frac{1}{12} \approx 0.083$. Related to the frequency is of course where in a cycle (i.e., in which position during a period/year a certain observations have). Consider the following example:

```
> (xseries <- ts(matrix(rep(2, 12)), start = c(1980,
+     3), frequency = 12))
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1980           2   2   2   2   2   2   2   2   2   2
1981   2   2
```

```
> cycle(xseries)
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1980           3   4   5   6   7   8   9  10  11  12
1981   1   2
```

That is, the time series `xseries` consists only of twelve 2's with starting date March 1980. It is monthly data since frequency is 12. The first observation has number 3 in the cycle (March being the third month of the year). The last observation (the 12'th) is second in the cycle since it has date February 1981.

Finally, we can extract the start and end dates of a time series object:

```
> start(cpi1)
```

```
[1] 1980    1
```

```
> end(cpi1)
```

```
[1] 2005   12
```

## 6 Selecting subsets of time series objects

We previously noted that although `cpi1` may be a time series object, say, `cpi1[221]` is not. So how can we define subsets of time series objects?[3] This is done with the function `window()`. The function takes `start` and `end` as arguments. Say we want to extract the period July 1987 to June 1989 from `cpi1`:

```
> window(cpi1, start = c(1987, 7), end = c(1988, 6))
```

```
       Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct
1987                                     166.9 167.8 169.4 170.1
1988 171.6 172.9 173.7 175.2 175.8 176.3
       Nov   Dec
1987 170.7 170.7
1988
```

If the argument `frequency` and/or `deltat` is provided, then the resulting time series is re-sampled at the new frequency given; else frequency is inherited from the original time series object. Hence,

[3] `subset()` cannot be used since it does not take time series objects as arguments.

```
> window(cpi1, start = c(1980, 2), deltat = 1)

Time Series:
Start = 1980.1
End = 2005.1
Frequency = 1
 [1]  96.8 109.3 119.0 128.8 138.9 151.0 159.0 164.4 172.9 184.0
[11] 199.9 225.0 230.3 241.6 245.9 252.3 255.8 254.2 256.6 256.3
[21] 258.7 262.6 269.4 278.4 277.3 279.2
```

re-samples data with frequency 1 starting with February 1980; i.e., the data from February each year will be extracted to constitute a new time series but now re-samples as yearly data (one observation per period/year).

We split the data into two time series objects:

```
> cpi1.a <- window(cpi1, end = c(1993, 12))
> cpi1.b <- window(cpi1, start = c(1992, 1))
```

The time series objects `cpi1.a` and `cpi1.b` now constitutes two different parts of the original time series object `cpi1`. They are, however, not mutually exclusive since data for the years 1992–1993 are in both. We can now combine these two objects using `ts.intersect()` and `ts.union()`:

```
> cpi1.c <- ts.union(cpi1.a, cpi1.b)
> cpi1.d <- ts.intersect(cpi1.a, cpi1.b)
```

We make the following observations regarding the created objects `cpi1.c` and `cpi1.d`:

- They are both time series objects containing two variables (`cpi1.a` and `cpi1.b`).
- `cpi1.c` is the union between `cpi1.a` and `cpi1.b` and contains 312 observations of the two variables. The first date is January 1980 and the last December 2005. There are a lot of missing values (`NA`):

  ```
  > str(cpi1.c)

   mts [1:312, 1:2] 95.3 96.8 97.2 97.9 98.2 ...
   - attr(*, "dimnames")=List of 2
    ..$ : NULL
    ..$ : chr [1:2] "cpi1.a" "cpi1.b"
   - attr(*, "tsp")= num [1:3] 1980 2006 12
   - attr(*, "class")= chr [1:2] "mts" "ts"

  > summary(cpi1.c)
  ```

```
     cpi1.a           cpi1.b
 Min.   : 95.3   Min.   :230
 1st Qu.:132.6   1st Qu.:255
 Median :163.3   Median :258
 Mean   :169.0   Mean   :260
 3rd Qu.:206.8   3rd Qu.:273
 Max.   :245.3   Max.   :282
 NA's   :144.0   NA's   :144
```

- `cpi1.d` is the intersection between `cpi1.a` and `cpi1.b` and contains only 24 observations of the two variables and where the variables are identical. The first date is January 1992 and last date December 1993. There are no missing values.

```
> str(cpi1.d)

 mts [1:24, 1:2] 230 230 231 232 232 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:2] "cpi1.a" "cpi1.b"
 - attr(*, "tsp")= num [1:3] 1992 1994 12
 - attr(*, "class")= chr [1:2] "mts" "ts"

> summary(cpi1.d)

     cpi1.a         cpi1.b
 Min.   :230   Min.   :230
 1st Qu.:232   1st Qu.:232
 Median :238   Median :238
 Mean   :238   Mean   :238
 3rd Qu.:243   3rd Qu.:243
 Max.   :245   Max.   :245
```

# 7 Lags and differences

In time series analysis observations from different dates, say $y_t$ from date $t$ and the preceding observation $y_{t-1}$ from date $t-1$, are often used in the same model. In terms of terminology and mathematical notation $y_{t-1}$ one usually call the (one period) lag of $y_t$. Sometimes a lag operator $\mathsf{L}$ is introduced, such that $\mathsf{L}\, y_t = y_{t-1}$. For longer lags, say lag $n$, one write $\mathsf{L}^n\, y_t = y_{t-n}$.

In R lags are created by the function `lag(y,k=1)`, where the default `k=1` implies a *lead. Therefore, note that the default value is equivalent to* $\mathsf{L}^{-1}\, y_t = y_{t+1}$! To get $\mathsf{L}\, y_t = y_{t-1}$ we must write `lag(y,k=-1)`. Consider the following example which implements $y_t$ and $\mathsf{L}\, y_t = y_{t-1}$:

```
> ts(1:5)

Time Series:
Start = 1
End = 5
Frequency = 1
[1] 1 2 3 4 5

> lag(ts(1:5), k = -1)

Time Series:
Start = 2
End = 6
Frequency = 1
[1] 1 2 3 4 5
```

In `ts(1:5)` the fifth observation with value 5 has date 5. On the other hand, in `lag(ts(1:5),k=-1` the fifth observation with value 5 has date 6. Therefore, only the time index is changed! The vector of observations is unaffected, which we see if we pick (say) the third observation of the time series and its lag:

```
> ts(1:5)[3]

[1] 3

> lag(ts(1:5), k = -1)[3]

[1] 3
```

This means that one has to careful using a time series and its lag in the same regression; see below.

R also has a function which directly defines differences between variables of different dates. This is the function `diff(y,lag=1,difference=1)`, with default arguments. Here the option `lag` has its intuitive meaning. The default values give the equivalent to `y-lag(y,k=-1)`:

```
> ts(1:5) - lag(ts(1:5), k = -1)

Time Series:
Start = 2
End = 5
Frequency = 1
[1] 1 1 1 1

> diff(ts(1:5), lag = 1, difference = 1)
```

```
Time Series:
Start = 2
End = 5
Frequency = 1
[1] 1 1 1 1
```

This means that `diff(y,lag=1,difference=1)` is exactly equivalent to the general difference operator $\Delta$, which is defined as $\Delta y_t = (1-\mathsf{L})y_t = y_t - y_{t-1}$.

However, one need also to be careful using `diff()`. For instance $y_t - y_{t-1}$ is implemented by

```
> diff(ts(1:5), lag = 2, difference = 1)

Time Series:
Start = 3
End = 5
Frequency = 1
[1] 2 2 2
```

which is replicate using `lag` as

```
> ts(1:5) - lag(ts(1:5), k = -2)

Time Series:
Start = 3
End = 5
Frequency = 1
[1] 2 2 2
```

On the other hand

```
> diff(ts(1:5), lag = 1, difference = 2)

Time Series:
Start = 3
End = 5
Frequency = 1
[1] 0 0 0
```

implements $\Delta^2 y_t = (1-\mathsf{L}^2)y_t = y_t - 2y_{t-1} + y_{t-2}$ which in turn is replicated by

```
> ts(1:5) - 2 * lag(ts(1:5), k = -1) + lag(ts(1:5),
+     k = -2)
```

```
Time Series:
Start = 3
End = 5
Frequency = 1
[1] 0 0 0
```

Finally, the function `diffinv()`. Given some vector of difference and an intimal value the undifferentiated series can be retrieved. Consider the first order difference $\Delta y_t = y_t - y_{t-1}$. Given $y_1$ and $\Delta y_t$ we can calculate $y_t$ using $y_t = y_{t-1} + \Delta y_t$ or

```
> diffinv(diff(ts(1:5), lag = 1, difference = 1), lag = 1,
+     difference = 1, 1)

Time Series:
Start = 1
End = 5
Frequency = 1
[1] 1 2 3 4 5
```

where the last 1 is $y_1$ (the first observation in the original series).

# 8 Regression and time series data

Let us define $y_t = \Delta \mathrm{CPI}_t$ and suppose that we want estimate the linear model $y_t = \alpha_0 + \alpha_1 y_{t-1}$. Note that the regression coefficient $\alpha_1$ in such model is identical to the coefficient of correlation between $y_t$ and $y_{t-1}$. If we plot `diff(cpi1)` against time we get an idea about the correlation:

```
> plot(diff(cpi1))
```

The correlation between the two series is accordingly very small. However, calculating the coefficient of correlation explicitly we get

```
> cor(diff(cpi1), lag(diff(cpi1), k = -1))

[1] 1
```

which is a result that we also get of we estimate the regression model with `lm()`:

```
> result <- lm(diff(cpi1) ~ lag(diff(cpi1), -1))
> summary(result)

Call:
lm(formula = diff(cpi1) ~ lag(diff(cpi1), -1))

Residuals:
      Min        1Q    Median        3Q       Max
-7.36e-15  1.00e-18  1.80e-17  3.70e-17  3.39e-16

Coefficients:
                     Estimate Std. Error   t value Pr(>|t|)
(Intercept)         -3.02e-16   2.73e-17 -1.11e+01   <2e-16 ***
lag(diff(cpi1), -1)  1.00e+00   2.14e-17  4.67e+16   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 4.25e-16 on 309 degrees of freedom
Multiple R-squared:    1,       Adjusted R-squared:    1
F-statistic: 2.18e+33 on 1 and 309 DF,  p-value: <2e-16
```

Despite that we can see in the graph that the correlation is low we get in both these instances that the coefficient of correlation is exactly unity.

The explanation is that `cor()` and `lm()` (as well as many other functions) are completely unaware of the time series properties of the data set. For instance, `lm()` takes as data argument a data frame or something that can be coerced to be a data frame. When our data is coerced to become a data frame is looses its time series properties:

```
> z <- data.frame(diff(cpi1), lag(diff(cpi1), -1))
> is.ts(z)

[1] FALSE
```

This can also be seen by looking at the first elements of `diff(cpi1)` and its lag by use of indices:

```
> diff(cpi1)[1:10]

 [1] 1.5 0.4 0.7 0.3 0.3 0.8 0.6 2.8 1.5 0.6

> lag(diff(cpi1), -1)[1:10]

 [1] 1.5 0.4 0.7 0.3 0.3 0.8 0.6 2.8 1.5 0.6
```

They are identical even though the observations refer to different dates. This means that the (standard) method of just imputing the mathematical expressions into the formula to be estimated does not work with the lag and difference operators.

Now, there are at least two methods to overcome this. The first method is to create a data set such that the variables get a common time index:

```
> library(tseries)
> y <- ts.union(diff(cpi1), lag(diff(cpi1), -1))
> y <- na.remove(y)
> is.ts(y[, 1])

[1] TRUE

> is.ts(y[, 2])

[1] TRUE
```

Note the now the separate columns of the time series object `y` are time series objects in themselves. This means that rows in the object have a common date. The library **tseries** is loaded to get access to the function `na.remove()`, which (as indicated by the name) removes missing values from time series objects. This accounts for the lower degrees of freedom below. Now the model is estimated

```
> result <- lm(y[, 1] ~ y[, 2])
> summary(result)

Call:
lm(formula = y[, 1] ~ y[, 2])

Residuals:
   Min     1Q Median     3Q    Max
-2.697 -0.707 -0.096  0.463  5.603

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.4947     0.0717    6.90  2.9e-11 ***
y[, 2]        0.1698     0.0561    3.03   0.0027 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.11 on 308 degrees of freedom
Multiple R-squared: 0.0289,       Adjusted R-squared: 0.0257
F-statistic: 9.16 on 1 and 308 DF,  p-value: 0.00269
```

Now it works because even if data with the common time index is coerced to a data frame its two vectors are not identical:

```
> y[1:10, 1]

 [1] 0.4 0.7 0.3 0.3 0.8 0.6 2.8 1.5 0.6 0.4

> y[1:10, 2]

 [1] 1.5 0.4 0.7 0.3 0.3 0.8 0.6 2.8 1.5 0.6
```

We can also use `cor()` to get the same result:

```
> cor(y[, 1], y[, 2])

[1] 0.1699
```

The second method is to invoke the library **dyn**. From the manual of the library we read:

> "**dyn** enables regression functions that were not written to handle time series to handle them. Both the dependent and independent variables may be time series and they may have different time indexes (in which case they are automatically aligned). The time series may also have missing values including internal missing values."

```
> library(dyn)
> result <- dyn$lm(diff(cpi1) ~ lag(diff(cpi1), -1))
> summary(result)

Call:
lm(formula = dyn(diff(cpi1) ~ lag(diff(cpi1), -1)))

Residuals:
   Min     1Q Median     3Q    Max
-2.697 -0.707 -0.096  0.463  5.603

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)           0.4947     0.0717    6.90  2.9e-11 ***
lag(diff(cpi1), -1)   0.1698     0.0561    3.03   0.0027 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.11 on 308 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared: 0.0289,       Adjusted R-squared: 0.0257
F-statistic: 9.16 on 1 and 308 DF,  p-value: 0.00269
```

We verify that the two methods give the same result; $\hat{\alpha}_1 = 0.17$. One difference, though, is that although in both cases we have 308 degrees of freedom we get using **dyn** a report of missing values (2 missing values in fact). The reason is that this is in relation to original (undifferentiated and unlagged) the time series object `cpi1`, which we have differenced (one missing value) and lagged (another missing value).

# 9 Time and seasonal dummy variables

Once we have defined a time series using `ts()` it is easy to define time and seasonal dummy variables. We start with the time dummy using `time()`:

```
> time(cpi1)
```

| | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug |
|---|---|---|---|---|---|---|---|---|
| 1980 | 1980.0 | 1980.1 | 1980.2 | 1980.2 | 1980.3 | 1980.4 | 1980.5 | 1980.6 |
| 1981 | 1981.0 | 1981.1 | 1981.2 | 1981.2 | 1981.3 | 1981.4 | 1981.5 | 1981.6 |
| 1982 | 1982.0 | 1982.1 | 1982.2 | 1982.2 | 1982.3 | 1982.4 | 1982.5 | 1982.6 |
| 1983 | 1983.0 | 1983.1 | 1983.2 | 1983.2 | 1983.3 | 1983.4 | 1983.5 | 1983.6 |
| 1984 | 1984.0 | 1984.1 | 1984.2 | 1984.2 | 1984.3 | 1984.4 | 1984.5 | 1984.6 |
| 1985 | 1985.0 | 1985.1 | 1985.2 | 1985.2 | 1985.3 | 1985.4 | 1985.5 | 1985.6 |
| 1986 | 1986.0 | 1986.1 | 1986.2 | 1986.2 | 1986.3 | 1986.4 | 1986.5 | 1986.6 |
| 1987 | 1987.0 | 1987.1 | 1987.2 | 1987.2 | 1987.3 | 1987.4 | 1987.5 | 1987.6 |
| 1988 | 1988.0 | 1988.1 | 1988.2 | 1988.2 | 1988.3 | 1988.4 | 1988.5 | 1988.6 |
| 1989 | 1989.0 | 1989.1 | 1989.2 | 1989.2 | 1989.3 | 1989.4 | 1989.5 | 1989.6 |
| 1990 | 1990.0 | 1990.1 | 1990.2 | 1990.2 | 1990.3 | 1990.4 | 1990.5 | 1990.6 |
| 1991 | 1991.0 | 1991.1 | 1991.2 | 1991.2 | 1991.3 | 1991.4 | 1991.5 | 1991.6 |
| 1992 | 1992.0 | 1992.1 | 1992.2 | 1992.2 | 1992.3 | 1992.4 | 1992.5 | 1992.6 |
| 1993 | 1993.0 | 1993.1 | 1993.2 | 1993.2 | 1993.3 | 1993.4 | 1993.5 | 1993.6 |
| 1994 | 1994.0 | 1994.1 | 1994.2 | 1994.2 | 1994.3 | 1994.4 | 1994.5 | 1994.6 |
| 1995 | 1995.0 | 1995.1 | 1995.2 | 1995.2 | 1995.3 | 1995.4 | 1995.5 | 1995.6 |
| 1996 | 1996.0 | 1996.1 | 1996.2 | 1996.2 | 1996.3 | 1996.4 | 1996.5 | 1996.6 |
| 1997 | 1997.0 | 1997.1 | 1997.2 | 1997.2 | 1997.3 | 1997.4 | 1997.5 | 1997.6 |
| 1998 | 1998.0 | 1998.1 | 1998.2 | 1998.2 | 1998.3 | 1998.4 | 1998.5 | 1998.6 |
| 1999 | 1999.0 | 1999.1 | 1999.2 | 1999.2 | 1999.3 | 1999.4 | 1999.5 | 1999.6 |
| 2000 | 2000.0 | 2000.1 | 2000.2 | 2000.2 | 2000.3 | 2000.4 | 2000.5 | 2000.6 |
| 2001 | 2001.0 | 2001.1 | 2001.2 | 2001.2 | 2001.3 | 2001.4 | 2001.5 | 2001.6 |
| 2002 | 2002.0 | 2002.1 | 2002.2 | 2002.2 | 2002.3 | 2002.4 | 2002.5 | 2002.6 |
| 2003 | 2003.0 | 2003.1 | 2003.2 | 2003.2 | 2003.3 | 2003.4 | 2003.5 | 2003.6 |
| 2004 | 2004.0 | 2004.1 | 2004.2 | 2004.2 | 2004.3 | 2004.4 | 2004.5 | 2004.6 |
| 2005 | 2005.0 | 2005.1 | 2005.2 | 2005.2 | 2005.3 | 2005.4 | 2005.5 | 2005.6 |

| | Sep | Oct | Nov | Dec |
|---|---|---|---|---|
| 1980 | 1980.7 | 1980.8 | 1980.8 | 1980.9 |
| 1981 | 1981.7 | 1981.8 | 1981.8 | 1981.9 |
| 1982 | 1982.7 | 1982.8 | 1982.8 | 1982.9 |
| 1983 | 1983.7 | 1983.8 | 1983.8 | 1983.9 |
| 1984 | 1984.7 | 1984.8 | 1984.8 | 1984.9 |
| 1985 | 1985.7 | 1985.8 | 1985.8 | 1985.9 |
| 1986 | 1986.7 | 1986.8 | 1986.8 | 1986.9 |
| 1987 | 1987.7 | 1987.8 | 1987.8 | 1987.9 |
| 1988 | 1988.7 | 1988.8 | 1988.8 | 1988.9 |
| 1989 | 1989.7 | 1989.8 | 1989.8 | 1989.9 |
| 1990 | 1990.7 | 1990.8 | 1990.8 | 1990.9 |
| 1991 | 1991.7 | 1991.8 | 1991.8 | 1991.9 |
| 1992 | 1992.7 | 1992.8 | 1992.8 | 1992.9 |
| 1993 | 1993.7 | 1993.8 | 1993.8 | 1993.9 |
| 1994 | 1994.7 | 1994.8 | 1994.8 | 1994.9 |
| 1995 | 1995.7 | 1995.8 | 1995.8 | 1995.9 |

```
1996 1996.7 1996.8 1996.8 1996.9
1997 1997.7 1997.8 1997.8 1997.9
1998 1998.7 1998.8 1998.8 1998.9
1999 1999.7 1999.8 1999.8 1999.9
2000 2000.7 2000.8 2000.8 2000.9
2001 2001.7 2001.8 2001.8 2001.9
2002 2002.7 2002.8 2002.8 2002.9
2003 2003.7 2003.8 2003.8 2003.9
2004 2004.7 2004.8 2004.8 2004.9
2005 2005.7 2005.8 2005.8 2005.9
```

Note that (i) the variable is a time series variable and (ii) that units are in increments of a year starting at 1980. However, since we most frequently use time variables to remove time trends units may matter. If units matter we can transform units to something appropriate. For example

```
> (TIME <- 1 + frequency(cpi1) * (time(cpi1) - start(cpi1)[1]))
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1980   1   2   3   4   5   6   7   8   9  10  11  12
1981  13  14  15  16  17  18  19  20  21  22  23  24
1982  25  26  27  28  29  30  31  32  33  34  35  36
1983  37  38  39  40  41  42  43  44  45  46  47  48
1984  49  50  51  52  53  54  55  56  57  58  59  60
1985  61  62  63  64  65  66  67  68  69  70  71  72
1986  73  74  75  76  77  78  79  80  81  82  83  84
1987  85  86  87  88  89  90  91  92  93  94  95  96
1988  97  98  99 100 101 102 103 104 105 106 107 108
1989 109 110 111 112 113 114 115 116 117 118 119 120
1990 121 122 123 124 125 126 127 128 129 130 131 132
1991 133 134 135 136 137 138 139 140 141 142 143 144
1992 145 146 147 148 149 150 151 152 153 154 155 156
1993 157 158 159 160 161 162 163 164 165 166 167 168
1994 169 170 171 172 173 174 175 176 177 178 179 180
1995 181 182 183 184 185 186 187 188 189 190 191 192
1996 193 194 195 196 197 198 199 200 201 202 203 204
1997 205 206 207 208 209 210 211 212 213 214 215 216
1998 217 218 219 220 221 222 223 224 225 226 227 228
1999 229 230 231 232 233 234 235 236 237 238 239 240
2000 241 242 243 244 245 246 247 248 249 250 251 252
2001 253 254 255 256 257 258 259 260 261 262 263 264
2002 265 266 267 268 269 270 271 272 273 274 275 276
2003 277 278 279 280 281 282 283 284 285 286 287 288
2004 289 290 291 292 293 294 295 296 297 298 299 300
2005 301 302 303 304 305 306 307 308 309 310 311 312
```

which also preserves the time series properties. Note that both these examples are independent of the length of the time series variable and its frequency, which can be useful in programming. Equivalent to the second example in all respects is the following:

```
> (TIME2 <- ts(1:length(cpi1), start = start(cpi1),
+     frequency = frequency(cpi1)))

     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1980   1   2   3   4   5   6   7   8   9  10  11  12
1981  13  14  15  16  17  18  19  20  21  22  23  24
1982  25  26  27  28  29  30  31  32  33  34  35  36
1983  37  38  39  40  41  42  43  44  45  46  47  48
1984  49  50  51  52  53  54  55  56  57  58  59  60
1985  61  62  63  64  65  66  67  68  69  70  71  72
1986  73  74  75  76  77  78  79  80  81  82  83  84
1987  85  86  87  88  89  90  91  92  93  94  95  96
1988  97  98  99 100 101 102 103 104 105 106 107 108
1989 109 110 111 112 113 114 115 116 117 118 119 120
1990 121 122 123 124 125 126 127 128 129 130 131 132
1991 133 134 135 136 137 138 139 140 141 142 143 144
1992 145 146 147 148 149 150 151 152 153 154 155 156
1993 157 158 159 160 161 162 163 164 165 166 167 168
1994 169 170 171 172 173 174 175 176 177 178 179 180
1995 181 182 183 184 185 186 187 188 189 190 191 192
1996 193 194 195 196 197 198 199 200 201 202 203 204
1997 205 206 207 208 209 210 211 212 213 214 215 216
1998 217 218 219 220 221 222 223 224 225 226 227 228
1999 229 230 231 232 233 234 235 236 237 238 239 240
2000 241 242 243 244 245 246 247 248 249 250 251 252
2001 253 254 255 256 257 258 259 260 261 262 263 264
2002 265 266 267 268 269 270 271 272 273 274 275 276
2003 277 278 279 280 281 282 283 284 285 286 287 288
2004 289 290 291 292 293 294 295 296 297 298 299 300
2005 301 302 303 304 305 306 307 308 309 310 311 312
```

All three can be used in regressions using both methods from the previous section:

```
> summary(dyn$lm(diff(cpi1) ~ lag(diff(cpi1), -1) +
+     time(cpi1)))

Call:
lm(formula = dyn(diff(cpi1) ~ lag(diff(cpi1), -1) + time(cpi1)))
```

```
Residuals:
   Min     1Q Median     3Q    Max
-2.556 -0.700 -0.177  0.433  5.507

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         63.78351   17.06438    3.74  0.00022 ***
lag(diff(cpi1), -1)  0.11918    0.05665    2.10  0.03621 *
time(cpi1)          -0.03174    0.00856   -3.71  0.00025 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.09 on 307 degrees of freedom
  (3 observations deleted due to missingness)
Multiple R-squared: 0.0705,        Adjusted R-squared: 0.0645
F-statistic: 11.6 on 2 and 307 DF,  p-value: 1.33e-05

> summary(dyn$lm(diff(cpi1) ~ lag(diff(cpi1), -1) +
+     TIME))

Call:
lm(formula = dyn(diff(cpi1) ~ lag(diff(cpi1), -1) + TIME))

Residuals:
   Min     1Q Median     3Q    Max
-2.556 -0.700 -0.177  0.433  5.507

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          0.941693   0.139484    6.75  7.3e-11 ***
lag(diff(cpi1), -1)  0.119183   0.056652    2.10  0.03621 *
TIME                -0.002645   0.000713   -3.71  0.00025 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.09 on 307 degrees of freedom
  (3 observations deleted due to missingness)
Multiple R-squared: 0.0705,        Adjusted R-squared: 0.0645
F-statistic: 11.6 on 2 and 307 DF,  p-value: 1.33e-05

> summary(dyn$lm(diff(cpi1) ~ lag(diff(cpi1), -1) +
+     TIME2))

Call:
lm(formula = dyn(diff(cpi1) ~ lag(diff(cpi1), -1) + TIME2))
```

```
Residuals:
   Min     1Q Median     3Q    Max 
-2.556 -0.700 -0.177  0.433  5.507 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)          0.941693   0.139484    6.75  7.3e-11 ***
lag(diff(cpi1), -1)  0.119183   0.056652    2.10  0.03621 *  
TIME2               -0.002645   0.000713   -3.71  0.00025 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 1.09 on 307 degrees of freedom
  (3 observations deleted due to missingness)
Multiple R-squared: 0.0705,	Adjusted R-squared: 0.0645 
F-statistic: 11.6 on 2 and 307 DF,  p-value: 1.33e-05 
```

In terms of inference they are equivalent, but (of course) the choice time units affect the size of regression coefficients and their estimated standard deviations.

Seasonal dummy variables are more complicated but nonetheless easier to construct. Basically, the number of seasons equals the frequency in the data and we therefore need `frequency` minus one seasonal dummy variables. We show only one method from package **forecast** which is really simple:

```
> library(forecast)

This is forecast 3.03 

> SEASON <- ts(seasonaldummy(diff(cpi1)), start = start(diff(cpi1)),
+     frequency = frequency(diff(cpi1)))
> summary(dyn$lm(diff(cpi1) ~ SEASON))

Call:
lm(formula = dyn(diff(cpi1) ~ SEASON))

Residuals:
   Min     1Q Median     3Q    Max 
-3.320 -0.485 -0.069  0.431  5.215 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   0.1692     0.1960    0.86  0.38850    
SEASON1       0.9508     0.2799    3.40  0.00077 ***
```

```
SEASON2        0.7154     0.2771    2.58  0.01032 *
SEASON3        0.9808     0.2771    3.54  0.00047 ***
SEASON4        0.6538     0.2771    2.36  0.01895 *
SEASON5        0.3385     0.2771    1.22  0.22293
SEASON6       -0.3154     0.2771   -1.14  0.25601
SEASON7       -0.1654     0.2771   -0.60  0.55110
SEASON8        0.0385     0.2771    0.14  0.88971
SEASON9        1.6000     0.2771    5.77  1.9e-08 ***
SEASON10       0.5308     0.2771    1.92  0.05641 .
SEASON11      -0.1423     0.2771   -0.51  0.60797
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.999 on 299 degrees of freedom
Multiple R-squared: 0.242,        Adjusted R-squared: 0.214
F-statistic: 8.66 on 11 and 299 DF,  p-value: 2.56e-13
```

Note that with monthly data only 11 variables are created. Note also that the number of observations is 311 due to the first order difference and for this reason the definition of the dummy variable is based on the first order difference.

# 10 Aggregation of time series data

Sometimes data are available as high frequency data but what you need is low frequency data. Say, you have monthly data but need quarterly or yearly data. The function `aggregate()` has a method for time series objects so that one can use `aggregate(x, nfrequency = 1, FUN = sum,...)`, where `x` is a time series object. The function splits `x` in blocks of length `frequency(x)/nfrequency`, where `nfrequency` is the frequency of the aggregated data (which must be lower than the frequency of `x` and which is 1 (yearly data) by default). The function specified by the argument `FUN` is then applied to each block of data (be default `sum`). For other arguments see `?aggregate`.

The default function `sum` may be suitable for flow data as (for example) the data set `AirPassengers`, which records the total number of international air passengers per month. Say we want to aggregate this data to quarterly data

```
> AirPassengers

     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
```

```
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

```
> (AirPassengers.Q <- aggregate(AirPassengers, nfrequency = 4))
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
1949  362  385  432  341
1950  382  409  498  387
1951  473  513  582  474
1952  544  582  681  557
1953  628  707  773  592
1954  627  725  854  661
1955  742  854 1023  789
1956  878 1005 1173  883
1957  972 1125 1336  988
1958 1020 1146 1400 1006
1959 1108 1288 1570 1174
1960 1227 1468 1736 1283
```

That is, the monthly observations are just summed quarter by quarter.

Note however, that you need monthly observations for "full" quarters for this result. Say data is incomplete in the sense that it starts in February:

```
> AirPassengers.N <- window(AirPassengers, start = c(1949,
+     2))
> (AirPassengers.N.Q <- aggregate(AirPassengers.N,
+     nfrequency = 4))
```

```
Time Series:
Start = 1949.08333333333
End = 1960.58333333333
Frequency = 4
 [1]  379  404  403  337  402  444  461  399  491  549  545  483
[13]  554  631  642  562  667  736  720  585  650  800  781  674
[25]  769  949  933  799  907 1105 1066  892 1005 1242 1218  981
[37] 1028 1289 1268 1007 1144 1440 1429 1184 1271 1629 1575
```

Aggregation will lead to quarters, but quarters consisting of February–April, May–July, August–October and November–January. The last two observations in the data set `AirPassengers.N`, November and December 1960, do not constitute a full quarter and are therefore dismissed.

If data are flows the default function `sum` is appropriate, but one may also have data in levels. Then (say) `mean` is an alternative in aggregation. The data set `nottem` contains monthly average air temperatures at Nottingham Castle in degrees Fahrenheit for the period 1920–1939. Say we want to construct quarterly averages:

```
> nottem

      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
1922 37.5 38.7 39.5 42.1 55.7 57.8 56.8 54.3 54.3 47.1 41.8 41.7
1923 41.8 40.1 42.9 45.8 49.2 52.7 64.2 59.6 54.4 49.2 36.3 37.6
1924 39.3 37.5 38.3 45.5 53.2 57.7 60.8 58.2 56.4 49.8 44.4 43.6
1925 40.0 40.5 40.8 45.1 53.8 59.4 63.5 61.0 53.0 50.0 38.1 36.3
1926 39.2 43.4 43.4 48.9 50.6 56.8 62.5 62.0 57.5 46.7 41.6 39.8
1927 39.4 38.5 45.3 47.1 51.7 55.0 60.4 60.5 54.7 50.3 42.3 35.2
1928 40.8 41.1 42.8 47.3 50.9 56.4 62.2 60.5 55.4 50.2 43.0 37.3
1929 34.8 31.3 41.0 43.9 53.1 56.9 62.5 60.3 59.8 49.2 42.9 41.9
1930 41.6 37.1 41.2 46.9 51.2 60.4 60.1 61.6 57.0 50.9 43.0 38.8
1931 37.1 38.4 38.4 46.5 53.5 58.4 60.6 58.2 53.8 46.6 45.5 40.6
1932 42.4 38.4 40.3 44.6 50.9 57.0 62.1 63.5 56.3 47.3 43.6 41.8
1933 36.2 39.3 44.5 48.7 54.2 60.8 65.5 64.9 60.1 50.2 42.1 35.8
1934 39.4 38.2 40.4 46.9 53.4 59.6 66.5 60.4 59.2 51.2 42.8 45.8
1935 40.0 42.6 43.5 47.1 50.0 60.5 64.6 64.0 56.8 48.6 44.2 36.4
1936 37.3 35.0 44.0 43.9 52.7 58.6 60.0 61.1 58.1 49.6 41.6 41.3
1937 40.8 41.0 38.4 47.4 54.1 58.6 61.4 61.8 56.3 50.9 41.4 37.1
1938 42.1 41.2 47.3 46.6 52.4 59.0 59.6 60.4 57.0 50.7 47.8 39.2
1939 39.4 40.9 42.4 47.8 52.4 58.0 60.7 61.8 58.2 46.7 46.6 37.8
```

```
> (nottem.Q <- aggregate(nottem, nfrequency = 4, FUN = mean))

       Qtr1   Qtr2   Qtr3   Qtr4
1920 41.933 53.100 56.133 44.400
1921 43.033 53.267 61.067 45.567
1922 38.567 51.867 55.133 43.533
1923 41.600 49.233 59.400 41.033
1924 38.367 52.133 58.467 45.933
1925 40.433 52.767 59.167 41.467
1926 42.000 52.100 60.667 42.700
1927 41.067 51.267 58.533 42.600
```

```
1928 41.567 51.533 59.367 43.500
1929 35.700 51.300 60.867 44.667
1930 39.967 52.833 59.567 44.233
1931 37.967 52.800 57.533 44.233
1932 40.367 50.833 60.633 44.233
1933 40.000 54.567 63.500 42.700
1934 39.333 53.300 62.033 46.600
1935 42.033 52.533 61.800 43.067
1936 38.767 51.733 59.733 44.167
1937 40.067 53.367 59.833 43.133
1938 43.533 52.667 59.000 45.900
1939 40.900 52.733 60.233 43.700
```

## 11 Irregularly spaced time series data

The time series data objects we have encountered so far are *regular* in the sense that the time passing from one observation to another is always the same. Some type of data (in many cases high frequency data such as data from financial markets) is irregular so that the time between observations is not always the same.

There are several packages that attempt to handle this problem, but here we will only have a brief look at one of them. This is package **zoo** (Z[eilies]'s Ordered Observations, after the original author's last name). Instead of a `ts` object we now create a `zoo` object. As before we need a vector or matrix with data. However, we cannot just set the start or end time of the date. Wee need to have an index along which the data are ordered.

We start by constructing such a `zoo` object. First we simulate some observations:

```
> set.seed(1069)
> (z1.data <- rnorm(10))

 [1] -1.191128 -0.092736 -0.064871 -0.717833 -0.953335 -1.360759
 [7]  0.814218  0.640844 -0.525930 -3.256695

> (z2.data <- rnorm(10))

 [1]  2.35276 -0.74599  0.56855  0.72583  0.67866  1.49532
 [7] -1.14575  2.12064  1.53274 -0.62578
```

Then we create an index along which these observations are ordered:

```
> (z1.index <- Sys.Date() - sample(1:20, size = 10))
```

```
 [1] "2011-09-03" "2011-09-05" "2011-09-07" "2011-09-04"
 [5] "2011-09-18" "2011-09-19" "2011-09-21" "2011-09-12"
 [9] "2011-09-14" "2011-09-17"
```

```
> (z2.index <- Sys.Date() - sample(1:20, size = 10))
```

```
 [1] "2011-09-15" "2011-09-09" "2011-09-02" "2011-09-03"
 [5] "2011-09-10" "2011-09-08" "2011-09-04" "2011-09-05"
 [9] "2011-09-17" "2011-09-11"
```

With the data and the index we can create the `zoo` object using the function `zoo(x, order.by)`, where `x` is the data and `order.by` is the index:

```
> library(zoo)
> (z1 <- zoo(z1.data, z1.index))

2011-09-03 2011-09-04 2011-09-05 2011-09-07 2011-09-12
 -1.191128  -0.717833  -0.092736  -0.064871   0.640844
2011-09-14 2011-09-17 2011-09-18 2011-09-19 2011-09-21
 -0.525930  -3.256695  -0.953335  -1.360759   0.814218
```

```
> (z2 <- zoo(z2.data, z2.index))

2011-09-02 2011-09-03 2011-09-04 2011-09-05 2011-09-08
   0.56855    0.72583   -1.14575    2.12064    1.49532
2011-09-09 2011-09-10 2011-09-11 2011-09-15 2011-09-17
  -0.74599    0.67866   -0.62578    2.35276    1.53274
```

In real applications, of course, both data and the index are most probably available to the researcher.

There are several standard functions that become equipped with methods for `zoo` objects, such as `plot` etc.

However, combining different `zoo` objects we cannot use `ts.intersect()`, `ts.union()` etc. Instead `merge()` has methods for `zoo` objects which can be used; see `?merge.zoo` for details. We can for instance create both the intersection and union of two `zoo` objects:

```
> merge(z1, z2)

                  z1       z2
2011-09-02        NA  0.56855
2011-09-03 -1.191128  0.72583
2011-09-04 -0.717833 -1.14575
2011-09-05 -0.092736  2.12064
2011-09-07 -0.064871       NA
2011-09-08        NA  1.49532
```

```
2011-09-09         NA -0.74599
2011-09-10         NA  0.67866
2011-09-11         NA -0.62578
2011-09-12  0.640844        NA
2011-09-14 -0.525930        NA
2011-09-15         NA  2.35276
2011-09-17 -3.256695  1.53274
2011-09-18 -0.953335        NA
2011-09-19 -1.360759        NA
2011-09-21  0.814218        NA
```

```
> merge(z1, z2, all = FALSE)
```

```
                  z1       z2
2011-09-03 -1.191128  0.72583
2011-09-04 -0.717833 -1.14575
2011-09-05 -0.092736  2.12064
2011-09-17 -3.256695  1.53274
```

In the case of creating the union missing values are imputed at appropriate places.

Sometimes one wants to transform an irregular time series to a regular. In the previous example we have data observed on an irregular daily basis. Let us take the series `z1` as an example. There are ten observations starting October 8, 2010 and ending October 26, 2010. If we create a regular daily series with observations for each day between October 8, 2010 and October 26, 2010 (inclusive) we will have 19 observations. In reality we have 10 so days with no observation must be filled with `NA`'s, which may later be replaced by some imputed value.

The first step to create a regular time series is to create the time index along which observations are ordered. We use the starting and ending dates of `z1` and create an index which is increased by one (day) in each step:

```
> Z1.index <- zoo(, seq(start(z1), end(z1), by = 1))
```

The length of the index is 19 and has (by definition) the start and end dates of `z1`. The second step is to merge this index with the irregular time series object `z1`:

```
> (Z1 <- merge(z1, Z1.index))
```

```
2011-09-03 2011-09-04 2011-09-05 2011-09-06 2011-09-07
 -1.191128  -0.717833  -0.092736         NA  -0.064871
2011-09-08 2011-09-09 2011-09-10 2011-09-11 2011-09-12
        NA         NA         NA         NA   0.640844
2011-09-13 2011-09-14 2011-09-15 2011-09-16 2011-09-17
```

```
        NA  -0.525930          NA          NA  -3.256695
2011-09-18 2011-09-19 2011-09-20 2011-09-21
 -0.953335  -1.360759          NA   0.814218
```

Merging fills days for which we have no observation with `NA`'s. Once this step is completed it may be a good idea to redefine the data series to a regular time series object using the `zooreg` class:

```
> (Z1 <- zoo(Z1, frequency = 365))

2011-09-03 2011-09-04 2011-09-05 2011-09-06 2011-09-07
 -1.191128  -0.717833  -0.092736          NA  -0.064871
2011-09-08 2011-09-09 2011-09-10 2011-09-11 2011-09-12
        NA          NA          NA          NA   0.640844
2011-09-13 2011-09-14 2011-09-15 2011-09-16 2011-09-17
        NA  -0.525930          NA          NA  -3.256695
2011-09-18 2011-09-19 2011-09-20 2011-09-21
 -0.953335  -1.360759          NA   0.814218
```

Package **zoo** comes with several alternatives to replace `NA`'s with some numerical value:

`na.approx()` This method replaces missing values with a linear approximation from surrounding observations using the function `approx()`. Example: Say we have three observations $1, 2, 10$. We want to create two additional observations between the first and the second and between the second and the third. A linear approximation would result in the series $1, 1.5, 2, 6, 10$:

```
> x <- 1:3
> y <- c(1, 2, 10)
> approx(x, y, n = 5)

$x
[1] 1.0 1.5 2.0 2.5 3.0

$y
[1]  1.0  1.5  2.0  6.0 10.0
```

Applied to `Z1` we get:

```
> na.approx(Z1)

2011-09-03 2011-09-04 2011-09-05 2011-09-06 2011-09-07
 -1.191128  -0.717833  -0.092736  -0.078804  -0.064871
2011-09-08 2011-09-09 2011-09-10 2011-09-11 2011-09-12
```

```
  0.076272   0.217415   0.358558   0.499701   0.640844
2011-09-13 2011-09-14 2011-09-15 2011-09-16 2011-09-17
  0.057457  -0.525930  -1.436185  -2.346440  -3.256695
2011-09-18 2011-09-19 2011-09-20 2011-09-21
 -0.953335  -1.360759  -0.273270   0.814218
```

`na.locf` Replaces each `NA` with the most recent non–`NA`. If the first observation is `NA` it is removed by default:

```
> na.locf(Z1)

2011-09-03 2011-09-04 2011-09-05 2011-09-06 2011-09-07
 -1.191128  -0.717833  -0.092736  -0.092736  -0.064871
2011-09-08 2011-09-09 2011-09-10 2011-09-11 2011-09-12
 -0.064871  -0.064871  -0.064871  -0.064871   0.640844
2011-09-13 2011-09-14 2011-09-15 2011-09-16 2011-09-17
  0.640844  -0.525930  -0.525930  -0.525930  -3.256695
2011-09-18 2011-09-19 2011-09-20 2011-09-21
 -0.953335  -1.360759  -1.360759   0.814218
```

`na.spline` Replaces `NA` with values from a monotone cubic function fitted to the data:

```
> na.spline(Z1)

2011-09-03 2011-09-04 2011-09-05 2011-09-06 2011-09-07
 -1.191128  -0.717833  -0.092736   0.044282  -0.064871
2011-09-08 2011-09-09 2011-09-10 2011-09-11 2011-09-12
 -0.031040   0.149314   0.383324   0.578123   0.640844
2011-09-13 2011-09-14 2011-09-15 2011-09-16 2011-09-17
  0.413057  -0.525930  -2.321044  -3.751878  -3.256695
2011-09-18 2011-09-19 2011-09-20 2011-09-21
 -0.953335  -1.360759  -1.843246   0.814218
```

## 12 Disaggregation of time series data

The procedure used to create regular time series data out of irregular time series data can also be used in the case one need to disaggregate data which are standard time series objects. That is, transform data from lower frequency to higher frequency.

Consider the data set `LakeHuron` which consists of yearly observations of the level of Lake Huron 1875–1972, which we want to disaggregate to quarterly data. To make it simple we just consider the period 1875-1884:

```
> (LakeHuron <- window(LakeHuron, end = 1884))
```

```
Time Series:
Start = 1875
End = 1884
Frequency = 1
 [1] 580.38 581.86 580.97 580.80 579.79 580.39 580.42 580.82
 [9] 581.40 581.32
```

We then create the the time index, where we now want to have an index with dates for each quarter:

```
> LH.index <- zoo(, seq(start(LakeHuron)[1], end(LakeHuron)[1],
+     by = 1/4))
```

```
> (LH <- merge(as.zoo(LakeHuron), LH.index))
```

```
1875(1) 1875(2) 1875(3) 1875(4) 1876(1) 1876(2) 1876(3) 1876(4)
 580.38      NA      NA      NA  581.86      NA      NA      NA
1877(1) 1877(2) 1877(3) 1877(4) 1878(1) 1878(2) 1878(3) 1878(4)
 580.97      NA      NA      NA  580.80      NA      NA      NA
1879(1) 1879(2) 1879(3) 1879(4) 1880(1) 1880(2) 1880(3) 1880(4)
 579.79      NA      NA      NA  580.39      NA      NA      NA
1881(1) 1881(2) 1881(3) 1881(4) 1882(1) 1882(2) 1882(3) 1882(4)
 580.42      NA      NA      NA  580.82      NA      NA      NA
1883(1) 1883(2) 1883(3) 1883(4) 1884(1)
 581.40      NA      NA      NA  581.32
```

The above methods can then be used to replace missing values:

```
> na.approx(LH)
```

```
1875(1) 1875(2) 1875(3) 1875(4) 1876(1) 1876(2) 1876(3) 1876(4)
 580.38  580.75  581.12  581.49  581.86  581.64  581.41  581.19
1877(1) 1877(2) 1877(3) 1877(4) 1878(1) 1878(2) 1878(3) 1878(4)
 580.97  580.93  580.88  580.84  580.80  580.55  580.29  580.04
1879(1) 1879(2) 1879(3) 1879(4) 1880(1) 1880(2) 1880(3) 1880(4)
 579.79  579.94  580.09  580.24  580.39  580.40  580.40  580.41
1881(1) 1881(2) 1881(3) 1881(4) 1882(1) 1882(2) 1882(3) 1882(4)
 580.42  580.52  580.62  580.72  580.82  580.97  581.11  581.25
1883(1) 1883(2) 1883(3) 1883(4) 1884(1)
 581.40  581.38  581.36  581.34  581.32
```

```
> na.locf(LH)
```

```
1875(1) 1875(2) 1875(3) 1875(4) 1876(1) 1876(2) 1876(3) 1876(4)
 580.38  580.38  580.38  580.38  581.86  581.86  581.86  581.86
1877(1) 1877(2) 1877(3) 1877(4) 1878(1) 1878(2) 1878(3) 1878(4)
```

```
 580.97  580.97  580.97  580.97  580.80  580.80  580.80  580.80
1879(1) 1879(2) 1879(3) 1879(4) 1880(1) 1880(2) 1880(3) 1880(4)
 579.79  579.79  579.79  579.79  580.39  580.39  580.39  580.39
1881(1) 1881(2) 1881(3) 1881(4) 1882(1) 1882(2) 1882(3) 1882(4)
 580.42  580.42  580.42  580.42  580.82  580.82  580.82  580.82
1883(1) 1883(2) 1883(3) 1883(4) 1884(1)
 581.40  581.40  581.40  581.40  581.32

> na.spline(LH)

1875(1) 1875(2) 1875(3) 1875(4) 1876(1) 1876(2) 1876(3) 1876(4)
 580.38  581.17  581.65  581.87  581.86  581.69  581.43  581.16
1877(1) 1877(2) 1877(3) 1877(4) 1878(1) 1878(2) 1878(3) 1878(4)
 580.97  580.90  580.91  580.90  580.80  580.56  580.24  579.95
1879(1) 1879(2) 1879(3) 1879(4) 1880(1) 1880(2) 1880(3) 1880(4)
 579.79  579.82  579.99  580.21  580.39  580.47  580.47  580.44
1881(1) 1881(2) 1881(3) 1881(4) 1882(1) 1882(2) 1882(3) 1882(4)
 580.42  580.46  580.55  580.67  580.82  580.98  581.14  581.28
1883(1) 1883(2) 1883(3) 1883(4) 1884(1)
 581.40  581.47  581.50  581.45  581.32
```

In this case data were in levels, in which case we know start and end points. However, special care has to be taken if data measures flows where we only know the aggregate change. Such methods are not treated here.