

# Groupwise computations and other utilities in the **doBy** package

Søren Højsgaard

doBy version 4.5-12 as of 2014-11-12

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Data used for illustration</b>	<b>1</b>
<b>3</b>	<b>Working with groupwise data</b>	<b>2</b>
3.1	The <b>summaryBy</b> function . . . . .	2
3.1.1	Basic usage . . . . .	2
3.1.2	Statistics on functions of data . . . . .	4
3.1.3	Copying variables out with the <b>id</b> argument . . . . .	5
3.1.4	Using <b>'.'</b> on the left hand side of a formula . . . . .	5
3.1.5	Using <b>'.'</b> on the right hand side of a formula . . . . .	6
3.1.6	Using <b>'1'</b> on the right hand side of the formula . . . . .	6
3.1.7	Preserving names of variables using <b>keep.names</b> . . . . .	6
3.2	The <b>orderBy</b> function . . . . .	6
3.3	The <b>splitBy</b> function . . . . .	7
3.4	The <b>sampleBy</b> function . . . . .	8
3.5	The <b>subsetBy</b> function . . . . .	8
3.6	The <b>transformBy</b> function . . . . .	8
3.7	The <b>lapplyBy</b> function . . . . .	8
3.8	The <b>scaleBy</b> function . . . . .	9
<b>4</b>	<b>Create By-functions on the fly</b>	<b>9</b>

<b>5</b>	<b>Miscellaneous</b>	<b>12</b>
5.1	Specialize . . . . .	12
5.2	The <code>firstobs()</code> / <code>lastobs()</code> function . . . . .	12
5.3	The <code>which.maxn()</code> and <code>which.minn()</code> functions . . . . .	13
5.4	Subsequences - <code>subSeq()</code> . . . . .	13
5.5	Recoding values of a vector - <code>recodeVar()</code> . . . . .	14
5.6	Renaming columns of a dataframe or matrix – <code>renameCol()</code> . . . . .	14
5.7	Time since an event - <code>timeSinceEvent()</code> . . . . .	15
5.8	Example: Using <code>subSeq()</code> and <code>timeSinceEvent()</code> . . . . .	19
<b>6</b>	<b>Acknowledgements</b>	<b>24</b>

## 1 Introduction

The doBy package contains a variety of utility functions. This working document describes some of these functions. The package originally grew out of a need to calculate groupwise summary statistics (much in the spirit of PROC SUMMARY of the SAS system), but today the package contains many different utilities.

## 2 Data used for illustration

The description of the doBy package is based on the following datasets.

**CO2 data** The CO2 data frame comes from an experiment on the cold tolerance of the grass species *Echinochloa crus-galli*. To limit the amount of output we modify names and levels of variables as follows

```
data(CO2)
CO2 <- transform(CO2, Treat=Treatment, Treatment=NULL)
levels(CO2$Treat) <- c("nchil","chil")
levels(CO2$Type) <- c("Que","Mis")
CO2 <- subset(CO2, Plant %in% c("Qn1", "Qc1", "Mn1", "Mc1"))
```

**Airquality data** The airquality dataset contains air quality measurements in New York, May to September 1973. The months are coded as

5,...,9. To limit the output we only consider data for two months:

```
airquality <- subset(airquality, Month %in% c(5,6))
```

**Dietox data** The dietox data are provided in the `doBy` package and result from a study of the effect of adding vitamin E and/or copper to the feed of slaughter pigs.

## 3 Working with groupwise data

### 3.1 The `summaryBy` function

The `summaryBy` function is used for calculating quantities like “the mean and variance of  $x$  and  $y$  for each combination of two factors  $A$  and  $B$ ”. Examples are based on the `C02` data.

#### 3.1.1 Basic usage

The mean and variance of `uptake` and `conc` for each value of `Plant` is obtained by:

```
myfun1 <- function(x){c(m=mean(x), v=var(x))}  
summaryBy( conc + uptake ~ Plant, data=C02, FUN=myfun1)
```

	Plant	conc.m	conc.v	uptake.m	uptake.v
1	Qn1	435	100950	33.23	67.48
2	Qc1	435	100950	29.97	69.47
3	Mn1	435	100950	26.40	75.59
4	Mc1	435	100950	18.00	16.96

Above `myfun1()` is a function that returns a vector of named values. Note that the values returned by the function has been named as `m` and `v`. An alternative specification is:

```
summaryBy( list(c("conc","uptake"), "Plant"), data=C02, FUN=myfun1)
```

	Plant	conc.m	conc.v	uptake.m	uptake.v
1	Qn1	435	100950	33.23	67.48
2	Qc1	435	100950	29.97	69.47
3	Mn1	435	100950	26.40	75.59
4	Mc1	435	100950	18.00	16.96

If the result of the function(s) are not named, then the names in the output data in general become less intuitive:

```
myfun2 <- function(x){c(mean(x), var(x))}
summaryBy( conc + uptake ~ Plant, data=C02, FUN=myfun2)
```

	Plant	conc.FUN1	conc.FUN2	uptake.FUN1	uptake.FUN2
1	Qn1	435	100950	33.23	67.48
2	Qc1	435	100950	29.97	69.47
3	Mn1	435	100950	26.40	75.59
4	Mc1	435	100950	18.00	16.96

Another usage is to specify a list of functions each of which returns a single value:

```
summaryBy( conc + uptake ~ Plant, data=C02, FUN=list( mean, var ) )
```

	Plant	conc.mean	uptake.mean	conc.var	uptake.var
1	Qn1	435	33.23	100950	67.48
2	Qc1	435	29.97	100950	69.47
3	Mn1	435	26.40	100950	75.59
4	Mc1	435	18.00	100950	16.96

Notice that if we specify a list of functions of which some returns a vector with more than one element, then the proper names are not retrieved:

```
summaryBy(uptake~Plant, data=C02, FUN=list( mean, var, myfun1 ))
```

	Plant	uptake.FUN1	uptake.FUN2	uptake.FUN3	uptake.FUN4
1	Qn1	33.23	67.48	33.23	67.48
2	Qc1	29.97	69.47	29.97	69.47
3	Mn1	26.40	75.59	26.40	75.59
4	Mc1	18.00	16.96	18.00	16.96

One can “hard code” the function names into the output as

```
summaryBy(uptake~Plant, data=C02, FUN=list( mean, var, myfun1 ),
          fun.names=c("mean","var","mm","vv"))
```

	Plant	uptake.mean	uptake.var	uptake.mm	uptake.vv
1	Qn1	33.23	67.48	33.23	67.48
2	Qc1	29.97	69.47	29.97	69.47
3	Mn1	26.40	75.59	26.40	75.59
4	Mc1	18.00	16.96	18.00	16.96

### 3.1.2 Statistics on functions of data

We may want to calculate the mean and variance for the logarithm of `uptake`, for `uptake+conc` (not likely to be a useful statistic) as well as for `uptake`

and `conc`. This can be achieved as:

```
summaryBy(log(uptake) + I(conc+uptake) + conc+uptake ~ Plant, data=C02,
          FUN=myfun1)
```

	Plant	log(uptake).m	log(uptake).v	conc + uptake.m	conc + uptake.v	conc.m
1	Qn1	3.467	0.10168	468.2	104747	435
2	Qc1	3.356	0.11873	465.0	105297	435
3	Mn1	3.209	0.17928	461.4	105642	435
4	Mc1	2.864	0.06874	453.0	103157	435

		conc.v	uptake.m	uptake.v
1	100950	33.23	67.48	
2	100950	29.97	69.47	
3	100950	26.40	75.59	
4	100950	18.00	16.96	

The names of the variables become involved with this. The user may control the names of the variables directly:

```
summaryBy(log(uptake) + I(conc+uptake) + conc + uptake ~ Plant, data=C02,
          FUN=myfun1, var.names=c("log.upt", "conc+upt", "conc", "upt"))
```

	Plant	log.upt.m	log.upt.v	conc+upt.m	conc+upt.v	conc.m	conc.v	upt.m	upt.v
1	Qn1	3.467	0.10168	468.2	104747	435	100950	33.23	67.48
2	Qc1	3.356	0.11873	465.0	105297	435	100950	29.97	69.47
3	Mn1	3.209	0.17928	461.4	105642	435	100950	26.40	75.59
4	Mc1	2.864	0.06874	453.0	103157	435	100950	18.00	16.96

If one does not want output variables to contain parentheses then setting `p2d=TRUE` causes the parentheses to be replaced by dots (“.”).

```
summaryBy(log(uptake)+I(conc+uptake)~Plant, data=C02, p2d=TRUE,
          FUN=myfun1)
```

	Plant	log.uptake..m	log.uptake..v	conc + uptake.m	conc + uptake.v
1	Qn1	3.467	0.10168	468.2	104747
2	Qc1	3.356	0.11873	465.0	105297
3	Mn1	3.209	0.17928	461.4	105642
4	Mc1	2.864	0.06874	453.0	103157

### 3.1.3 Copying variables out with the `id` argument

To get the value of the `Type` and `Treat` in the first row of the groups (defined by the values of `Plant`) copied to the output dataframe we use the `id` argument in one of the following forms:

```
summaryBy(conc+uptake~Plant, data=C02, FUN=myfun1, id=~Type+Treat)
```

```

Plant conc.m conc.v uptake.m uptake.v Type Treat
1  Qn1    435 100950    33.23    67.48  Que nchil
2  Qc1    435 100950    29.97    69.47  Que  chil
3  Mn1    435 100950    26.40    75.59  Mis nchil
4  Mc1    435 100950    18.00    16.96  Mis  chil

summaryBy(conc+uptake~Plant, data=C02, FUN=myfun1, id=c("Type","Treat"))

Plant conc.m conc.v uptake.m uptake.v Type Treat
1  Qn1    435 100950    33.23    67.48  Que nchil
2  Qc1    435 100950    29.97    69.47  Que  chil
3  Mn1    435 100950    26.40    75.59  Mis nchil
4  Mc1    435 100950    18.00    16.96  Mis  chil

```

### 3.1.4 Using '.' on the left hand side of a formula

It is possible to use the dot (".") on the left hand side of the formula. The dot means "all numerical variables which do not appear elsewhere" (i.e. on the right hand side of the formula and in the id statement):

```

summaryBy(log(uptake)+I(conc+uptake)+. ~Plant, data=C02, FUN=myfun1)

Plant log(uptake).m log(uptake).v conc + uptake.m conc + uptake.v conc.m
1  Qn1           3.467           0.10168           468.2           104747           435
2  Qc1           3.356           0.11873           465.0           105297           435
3  Mn1           3.209           0.17928           461.4           105642           435
4  Mc1           2.864           0.06874           453.0           103157           435

conc.v uptake.m uptake.v
1 100950    33.23    67.48
2 100950    29.97    69.47
3 100950    26.40    75.59
4 100950    18.00    16.96

```

### 3.1.5 Using '.' on the right hand side of a formula

The dot (".") can also be used on the right hand side of the formula where it refers to "all non-numerical variables which are not specified elsewhere":

```

summaryBy(log(uptake) ~Plant+., data=C02, FUN=myfun1)

Plant Type Treat log(uptake).m log(uptake).v
1  Qn1  Que nchil           3.467           0.10168
2  Qc1  Que  chil           3.356           0.11873
3  Mn1  Mis nchil           3.209           0.17928
4  Mc1  Mis  chil           2.864           0.06874

```

### 3.1.6 Using '1' on the right hand side of the formula

Using 1 on the right hand side means no grouping:

```
summaryBy(log(uptake) ~ 1, data=C02, FUN=myfun1)
      log(uptake).m log(uptake).v
1           3.224           0.1577
```

### 3.1.7 Preserving names of variables using keep.names

If the function applied to data only returns one value, it is possible to force that the summary variables retain the original names by setting `keep.names=TRUE`. A typical use of this could be

```
summaryBy(conc+uptake+log(uptake)~Plant,
  data=C02, FUN=mean, id=~Type+Treat, keep.names=TRUE)
      Plant conc uptake log(uptake) Type Treat
1   Qn1  435  33.23           3.467  Que nchil
2   Qc1  435  29.97           3.356  Que  chil
3   Mn1  435  26.40           3.209  Mis nchil
4   Mc1  435  18.00           2.864  Mis  chil
```

## 3.2 The orderBy function

Ordering (or sorting) a data frame is possible with the `orderBy` function. Suppose we want to order the rows of the `airquality` data by `Temp` and by `Month` (within `Temp`). This can be achieved by:

```
x<-orderBy(~Temp+Month, data=airquality)
```

The first lines of the result are:

```
head(x)
      Ozone Solar.R Wind Temp Month Day
5      NA      NA 14.3   56     5    5
18     6      78 18.4   57     5   18
25     NA      66 16.6   57     5   25
27     NA      NA  8.0   57     5   27
15    18      65 13.2   58     5   15
26     NA     266 14.9   58     5   26
```

If we want the ordering to be by decreasing values of one of the variables, we change the sign, e.g.

```
x<-orderBy(~Temp+Month, data=airquality)
head(x)
```

	Ozone	Solar.R	Wind	Temp	Month	Day
42	NA	259	10.9	93	6	11
43	NA	250	9.2	92	6	12
40	71	291	13.8	90	6	9
39	NA	273	6.9	87	6	8
41	39	323	11.5	87	6	10
36	NA	220	8.6	85	6	5

### 3.3 The splitBy function

Suppose we want to split the `airquality` data into a list of dataframes, e.g. one dataframe for each month. This can be achieved by:

```
x<-splitBy(~Month, data=airquality)
x
```

	listentry	Month
1	5	5
2	6	6

Hence for month 5, the relevant entry-name in the list is '5' and this part of data can be extracted as

```
x[['5']]
```

Information about the grouping is stored as a dataframe in an attribute called `groupid` and can be retrieved with:

```
attr(x,"groupid")
```

	Month
1	5
2	6

### 3.4 The sampleBy function

Suppose we want a random sample of 50 % of the observations from a dataframe. This can be achieved with:

```
sampleBy(~1, frac=0.5, data=airquality)
```

Suppose instead that we want a systematic sample of every fifth observation within each month. This is achieved with:



```
sampleBy(~Month, frac=0.2, data=airquality, systematic=T)
```

### 3.5 The subsetBy function

Suppose we want to select those rows within each month for which the wind speed is larger than the mean wind speed (within the month). This is achieved by:

```
subsetBy(~Month, subset=Wind>mean(Wind), data=airquality)
```

Note that the statement `Wind>mean(Wind)` is evaluated within each month.

### 3.6 The transformBy function

The `transformBy` function is analogous to the `transform` function except that it works within groups. For example:

```
transformBy(~Month, data=airquality, minW=min(Wind), maxW=max(Wind),  
           chg=sum(range(Wind)*c(-1,1)))
```

### 3.7 The lapplyBy function

This `lapplyBy` function is a wrapper for first splitting data into a list according to the formula (using `splitBy`) and then applying a function to each element of the list (using `apply`).

Suppose we want to calculate the weekwise feed efficiency of the pigs in the `dietox` data, i.e. weight gain divided by feed intake.

```
data(dietox)  
dietox <- orderBy(~Pig+Time, data=dietox)  
FEfun  <- function(d){c(NA, diff(d$Weight)/diff(d$Feed))}  
v      <- lapplyBy(~Pig, data=dietox, FEfun)  
dietox$FE <- unlist(v)
```

Technically, the above is the same as

```
dietox <- orderBy(~Pig+Time, data=dietox)  
wdata  <- splitBy(~Pig, data=dietox)  
v      <- lapply(wdata, FEfun)  
dietox$FE <- unlist(v)
```

### 3.8 The scaleBy function

Standardize the iris data within each value of "Species":

```
x<-scaleBy( list(c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"),
                  "Species"), data=iris)
head(x)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	0.26667	0.1899	-0.3570	-0.4365	setosa
2	-0.30072	-1.1291	-0.3570	-0.4365	setosa
3	-0.86811	-0.6015	-0.9328	-0.4365	setosa
4	-1.15181	-0.8653	0.2188	-0.4365	setosa
5	-0.01702	0.4537	-0.3570	-0.4365	setosa
6	1.11776	1.2452	1.3705	1.4613	setosa

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

## 4 Create By-functions on the fly

Create a function for creating groupwise t-tests

```
mydata <- data.frame(y=rnorm(32), x=rnorm(32),
  g1=factor(rep(c(1,2),each=16)), g2=factor(rep(c(1,2), each=8)),
  g3=factor(rep(c(1,2),each=4)))
head(mydata)
```

	y	x	g1	g2	g3
1	0.096322	0.7948	1	1	1
2	0.546704	1.6288	1	1	1
3	-0.445704	0.1445	1	1	1
4	-1.342316	-0.4882	1	1	1
5	0.008648	0.8240	1	1	2
6	-0.454060	-0.1821	1	1	2

```
## Based on the formula interface to t.test
t.testBy1 <- function(formula, group, data, ...){
  formulaFunBy(formula, group, data, FUN=t.test, class="t.testBy1", ...)
}
## Based on the default interface to t.test
t.testBy2 <- function(formula, group, data, ...){
  xyFunBy(formula, group, data, FUN=t.test, class="t.testBy1", ...)
}
```

Notice: The optional `class` argument will facilitate that you create your own print / summary methods etc.

```
t.testBy1(y~g1, ~g2, data=mydata)
$`1`
```

Welch Two Sample t-test

```
data: y by g1
t = 0.9369, df = 13.45, p-value = 0.3654
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5982  1.5197
sample estimates:
mean in group 1 mean in group 2
 -0.06707      -0.52785
```

```
$`2`
```

Welch Two Sample t-test

```
data: y by g1
t = -0.9935, df = 13.18, p-value = 0.3383
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.0757  0.7667
sample estimates:
mean in group 1 mean in group 2
 -0.4391      0.2154
```

```
attr("class")
[1] "t.testBy1"
```

```

t.testBy2(y~x, ~g2, data=mydata)
$`1`

      Welch Two Sample t-test

data:  x and y
t = -2.523, df = 25.71, p-value = 0.01817
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.336 -0.136
sample estimates:
mean of x mean of y
 -0.2975    0.4388

$`2`

      Welch Two Sample t-test

data:  x and y
t = -0.7707, df = 22.68, p-value = 0.4489
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.0563  0.4832
sample estimates:
mean of x mean of y
 -0.1119    0.1747

attr(,"class")
[1] "t.testBy1"

```

## 5 Miscellaneous

### 5.1 Specialize

```

ff <- function(a,b=2,c=4){a+b+c}
ff1 <- specialize(ff, arglist=list(a=1, b=7, yy=123))
ff1

```

```

function (c = 4)
{
  1 + 7 + c
}
<environment: 0x00000000c3b1d08>

gg  <- rnorm
gg1 <- specialize(gg, list(n=10))
gg1

function (mean = 0, sd = 1)
.External(C_rnorm, 10, mean, sd)
<environment: 0x00000000c343388>

Notice that this result is absurd:
f  <- function(a) {a <- a + 1; a}
f1 <- specialize(f, list(a = 10))
f1

function ()
{
  10 <- 10 + 1
  10
}
<environment: 0x00000000c0d75b0>

```

## 5.2 The firstobs() / lastobs() function

To obtain the indices of the first/last occurrences of an item in a vector do:

```

x <- c(1,1,1,2,2,2,1,1,1,3)
firstobs(x)
[1]  1  4 10

lastobs(x)
[1]  6  9 10

```

The same can be done on a data frame, e.g.

```

firstobs(~Plant, data=C02)
[1]  1  8 15 22

lastobs(~Plant, data=C02)
[1]  7 14 21 28

```

### 5.3 The `which.maxn()` and `which.minn()` functions

The location of the  $n$  largest / smallest entries in a numeric vector can be obtained with

```
x <- c(1:4,0:5,11,NA,NA)
which.maxn(x,3)
[1] 11 10 4
which.minn(x,5)
[1] 5 1 6 2 7
```

### 5.4 Subsequences - `subSeq()`

Find (sub) sequences in a vector:

```
x <- c(1,1,2,2,2,1,1,3,3,3,3,1,1,1)
subSeq(x)
  first last length midpoint value
1     1     2      2        2     1
2     3     5      3        4     2
3     6     7      2        7     1
4     8    11      4       10     3
5    12    14      3       13     1
subSeq(x, item=1)
  first last length midpoint value
1     1     2      2        2     1
2     6     7      2        7     1
3    12    14      3       13     1
subSeq(letters[x])
  first last length midpoint value
1     1     2      2        2     a
2     3     5      3        4     b
3     6     7      2        7     a
4     8    11      4       10     c
5    12    14      3       13     a
subSeq(letters[x],item="a")
  first last length midpoint value
1     1     2      2        2     a
2     6     7      2        7     a
3    12    14      3       13     a
```

## 5.5 Recoding values of a vector - recodeVar()

```
x <- c("dec","jan","feb","mar","apr","may")
src1 <- list(c("dec","jan","feb"), c("mar","apr","may"))
tgt1 <- list("winter","spring")
recodeVar(x,src=src1,tgt=tgt1)

[1] "winter" "winter" "winter" "spring" "spring" "spring"
```

## 5.6 Renaming columns of a dataframe or matrix – renameCol()

```
head(renameCol(CO2, 1:2, c("kk","ll")))

  kk  ll conc uptake Treat
1 Qn1 Que   95   16.0 nchil
2 Qn1 Que  175   30.4 nchil
3 Qn1 Que  250   34.8 nchil
4 Qn1 Que  350   37.2 nchil
5 Qn1 Que  500   35.3 nchil
6 Qn1 Que  675   39.2 nchil

head(renameCol(CO2, c("Plant","Type"), c("kk","ll")))

  kk  ll conc uptake Treat
1 Qn1 Que   95   16.0 nchil
2 Qn1 Que  175   30.4 nchil
3 Qn1 Que  250   34.8 nchil
4 Qn1 Que  350   37.2 nchil
5 Qn1 Que  500   35.3 nchil
6 Qn1 Que  675   39.2 nchil
```

## 5.7 Time since an event - timeSinceEvent()

Consider the vector

```
yvar <- c(0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0)
```

Imagine that "1" indicates an event of some kind which takes place at a certain time point. By default time points are assumed equidistant but for illustration we define time time variable

```
tvar <- seq_along(yvar) + c(0.1,0.2)
```

Now we find time since event as

```
tse<- timeSinceEvent(yvar,tvar)
```

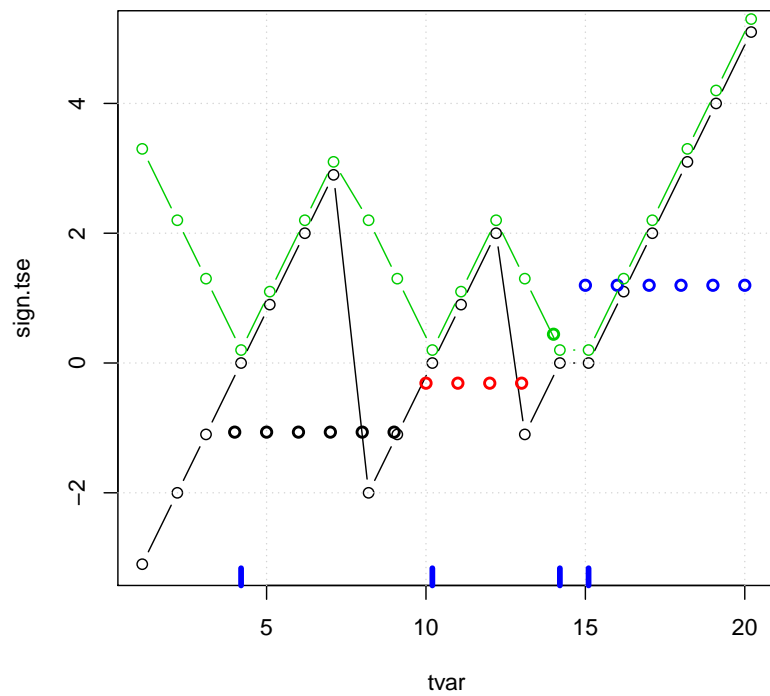
	yvar	tvar	abs.tse	sign.tse	ewin	run	tae	tbe
1	0	1.1	3.1	-3.1	1	NA	NA	-3.1
2	0	2.2	2.0	-2.0	1	NA	NA	-2.0
3	0	3.1	1.1	-1.1	1	NA	NA	-1.1
4	1	4.2	0.0	0.0	1	1	0.0	0.0
5	0	5.1	0.9	0.9	1	1	0.9	-5.1
6	0	6.2	2.0	2.0	1	1	2.0	-4.0
7	0	7.1	2.9	2.9	1	1	2.9	-3.1
8	0	8.2	2.0	-2.0	2	1	4.0	-2.0
9	0	9.1	1.1	-1.1	2	1	4.9	-1.1
10	1	10.2	0.0	0.0	2	2	0.0	0.0
11	0	11.1	0.9	0.9	2	2	0.9	-3.1
12	0	12.2	2.0	2.0	2	2	2.0	-2.0
13	0	13.1	1.1	-1.1	3	2	2.9	-1.1
14	1	14.2	0.0	0.0	3	3	0.0	0.0
15	1	15.1	0.0	0.0	4	4	0.0	0.0
16	0	16.2	1.1	1.1	4	4	1.1	NA
17	0	17.1	2.0	2.0	4	4	2.0	NA
18	0	18.2	3.1	3.1	4	4	3.1	NA
19	0	19.1	4.0	4.0	4	4	4.0	NA
20	0	20.2	5.1	5.1	4	4	5.1	NA

The output reads as follows:

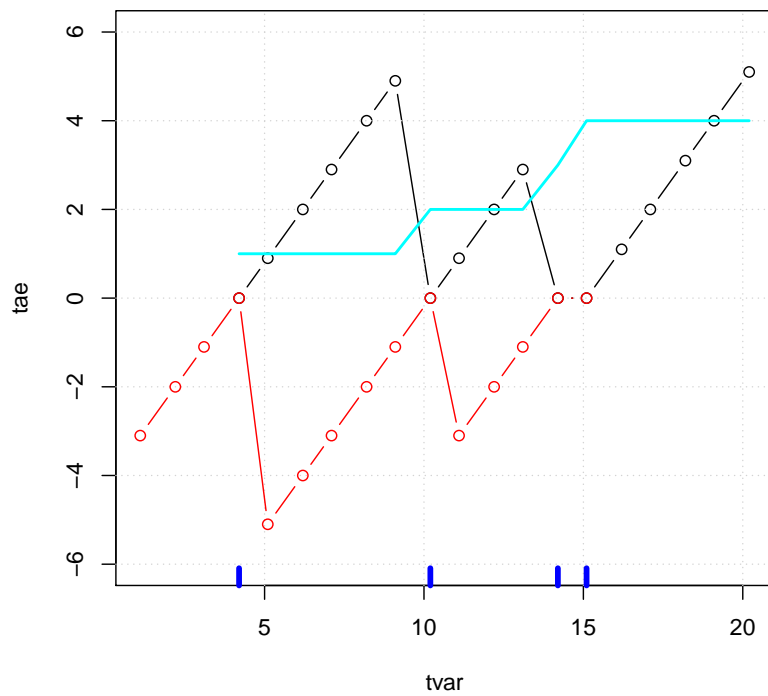
- **abs.tse**: Absolute time since (nearest) event.
- **sign.tse**: Signed time since (nearest) event.
- **ewin**: Event window: Gives a symmetric window around each event.
- **run**: The value of **run** is set to 1 when the first event occurs and is increased by 1 at each subsequent event.
- **tae**: Time after event.
- **tbe**: Time before event.

```
plot(sign.tse~tvar, data=tse, type="b")
grid()
rug(tse$tvar[tse$yvar==1], col='blue',lwd=4)
points(scale(tse$run), col=tse$run, lwd=2)
lines(abs.tse+.2~tvar, data=tse, type="b",col=3)
```

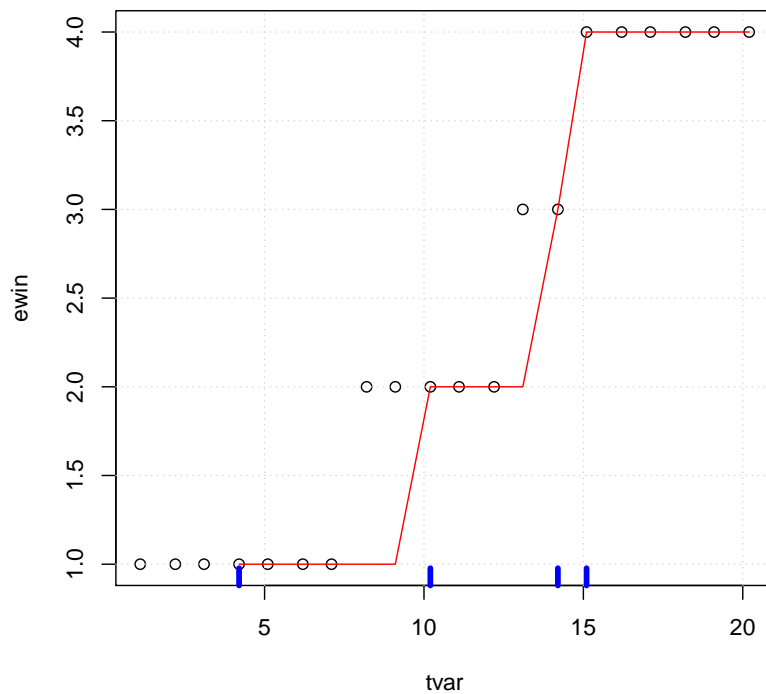




```
plot(tae~tvar, data=tse, ylim=c(-6,6),type="b")
grid()
lines(tbe~tvar, data=tse, type="b", col='red')
rug(tse$tvar[tse$yvar==1], col='blue',lwd=4)
lines(run~tvar, data=tse, col='cyan',lwd=2)
```



```
plot(ewin~tvar, data=tse,ylim=c(1,4))
rug(tse$tvar[tse$yvar==1], col='blue',lwd=4)
grid()
lines(run~tvar, data=tse,col='red')
```



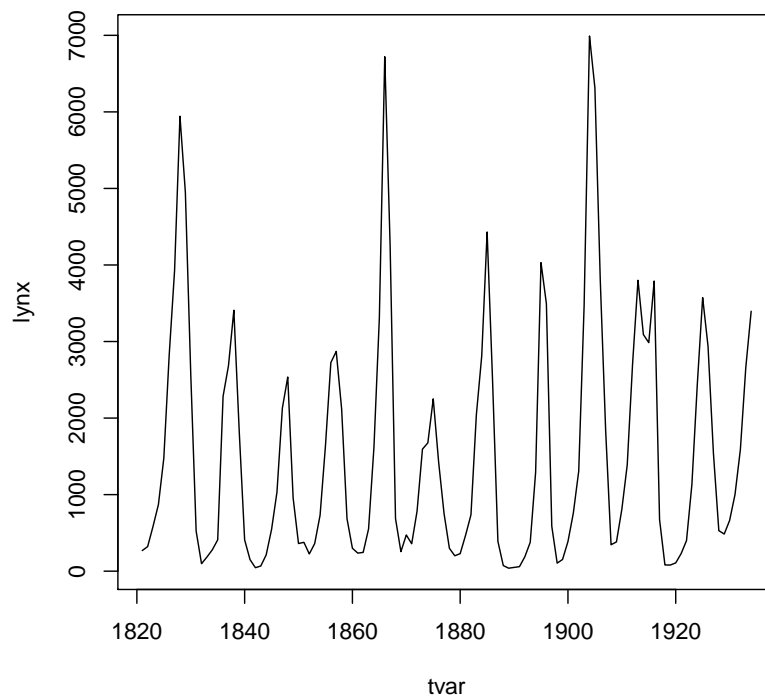
We may now find times for which time since an event is at most 1 as

```
tse$tvar[tse$abs<=1]
[1]  4.2  5.1 10.2 11.1 14.2 15.1
```

## 5.8 Example: Using subSeq() and timeSinceEvent()

Consider the lynx data:

```
lynx <- as.numeric(lynx)
tvar <- 1821:1934
plot(tvar,lynx,type='l')
```



Suppose we want to estimate the cycle lengths. One way of doing this is as follows:

```

yyy <- lynx>mean(lynx)
head(yyy)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE

sss <- subSeq(yyy,TRUE)
sss

```

	first	last	length	midpoint	value
1	6	10	5	8	TRUE
2	16	19	4	18	TRUE
3	27	28	2	28	TRUE
4	35	38	4	37	TRUE
5	44	47	4	46	TRUE
6	53	55	3	54	TRUE
7	63	66	4	65	TRUE
8	75	76	2	76	TRUE

```

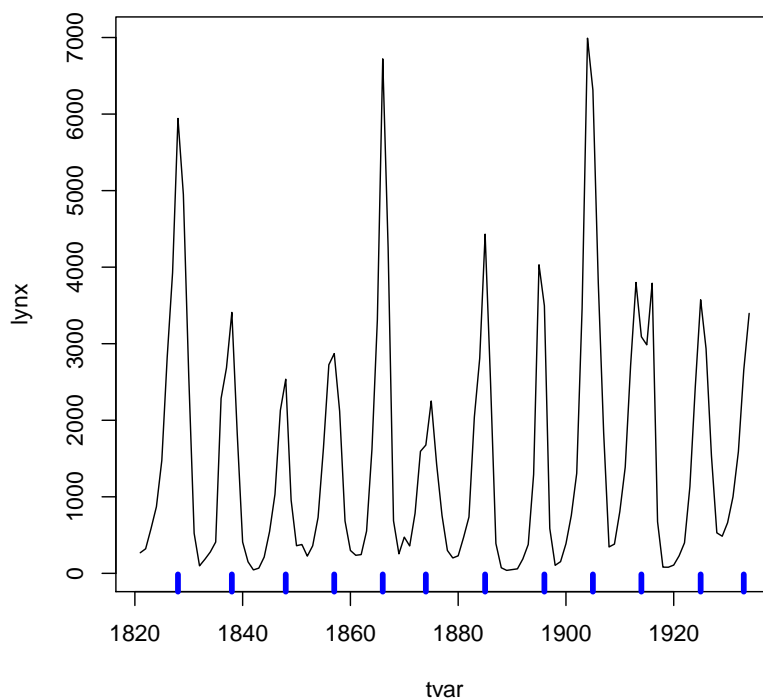
9      83   87      5      85  TRUE
10     92   96      5      94  TRUE
11    104  106      3     105  TRUE
12    112  114      3     113  TRUE

```

```

plot(tvar,lynx,type='l')
rug(tvar[sss$midpoint],col='blue',lwd=4)

```



Create the 'event vector'

```

yvar <- rep(0,length(lynx))
yvar[sss$midpoint] <- 1
str(yvar)

num [1:114] 0 0 0 0 0 0 0 0 1 0 0 ...

tse <- timeSinceEvent(yvar,tvar)
head(tse,20)

  yvar tvar abs.tse sign.tse ewin run tae tbe
1     0 1821      7      -7    1  NA  NA  -7

```

2	0	1822	6	-6	1	NA	NA	-6
3	0	1823	5	-5	1	NA	NA	-5
4	0	1824	4	-4	1	NA	NA	-4
5	0	1825	3	-3	1	NA	NA	-3
6	0	1826	2	-2	1	NA	NA	-2
7	0	1827	1	-1	1	NA	NA	-1
8	1	1828	0	0	1	1	0	0
9	0	1829	1	1	1	1	1	-9
10	0	1830	2	2	1	1	2	-8
11	0	1831	3	3	1	1	3	-7
12	0	1832	4	4	1	1	4	-6
13	0	1833	5	5	1	1	5	-5
14	0	1834	4	-4	2	1	6	-4
15	0	1835	3	-3	2	1	7	-3
16	0	1836	2	-2	2	1	8	-2
17	0	1837	1	-1	2	1	9	-1
18	1	1838	0	0	2	2	0	0
19	0	1839	1	1	2	2	1	-9
20	0	1840	2	2	2	2	2	-8

We get two different (not that different) estimates of period lengths:

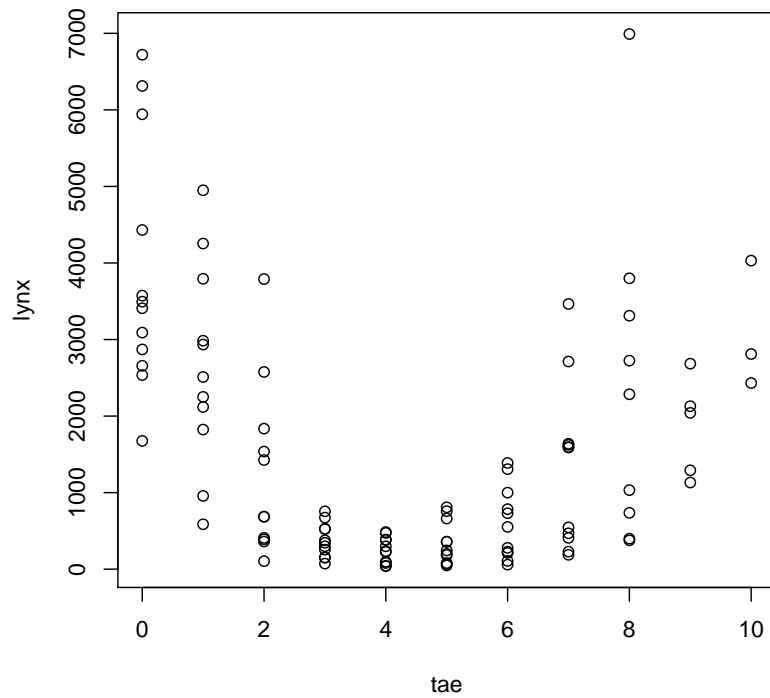
```
len1 <- tapply(tse$ewin, tse$ewin, length)
 1  2  3  4  5  6  7  8  9 10 11 12
13 10  9  9  9  9 11 10  9 10 10  5

len2 <- tapply(tse$run, tse$run, length)
 1  2  3  4  5  6  7  8  9 10 11 12
10 10  9  9  8 11 11  9  9 11  8  2

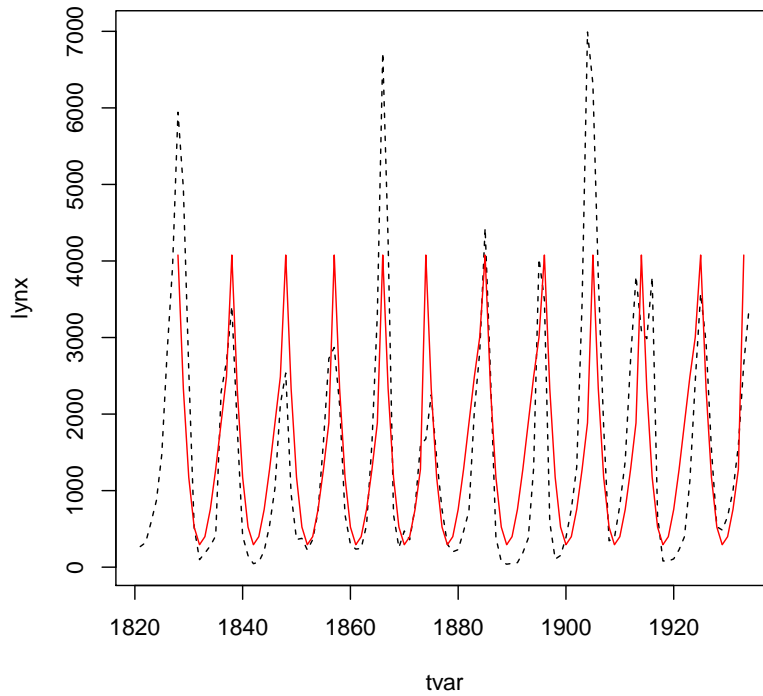
c(median(len1),median(len2),mean(len1),mean(len2))
[1] 9.500 9.000 9.500 8.917
```

We can overlay the cycles as:

```
tse$lynx <- lynx
tse2 <- na.omit(tse)
plot(lynx~tae, data=tse2)
```



```
plot(tvar,lynx,type='l',lty=2)
mm <- lm(lynx~tae+I(tae^2)+I(tae^3), data=tse2)
lines(fitted(mm)~tvar, data=tse2, col='red')
```



## 6 Acknowledgements

Credit is due to Dennis Chabot, Gabor Grothendieck, Paul Murrell, Jim Robison-Cox and Erik Jørgensen for reporting various bugs and making various suggestions to the functionality in the doBy package.