Introduction to bfast

Have a look at the dataset.

- the original (evio column) and the filtered one (evi) time series.
- the timeline in dates column.
- the registered date (dateevent) of an event related to the sample.
- the class (dataset column) that has been assigned to the sample at the event time.

Learn about 'ts', 'xts' and 'zoo' time series types. Try the 'xts' and 'zoo' R package. Construct a time series.

'ts' is a function for building time series objects in R. 'xts' is an R package offering a number of functionalities to work on time-indexed data. 'xts' extends zoo, another popular package for time-series analysis. The main benefit of xts is its seamless compatibility with other packages using different timeseries classes (timeSeries, zoo, ...). In addition 'xts' allows the user to add custom attributes to any object. 'zoo' is an R package that can handle irregular time series. 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.

Construct the first record of the file (the second line of your data) a time series (e.g. with meanevi)

```
library(zoo)
```

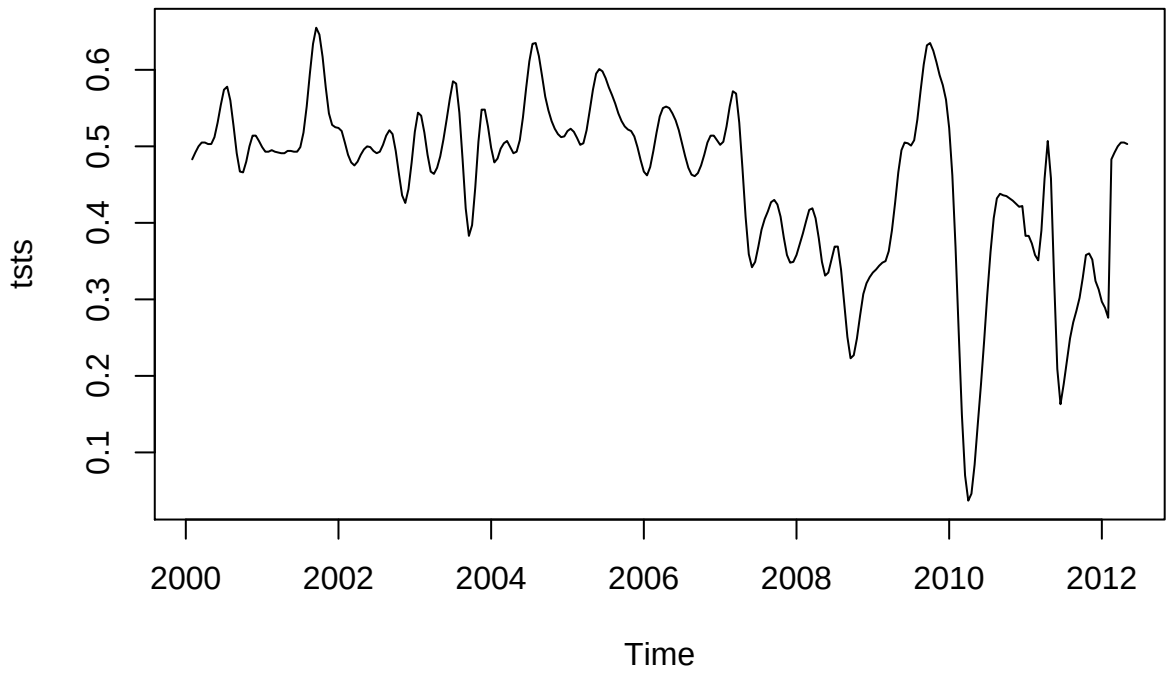
```
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
deter = read.csv("/home/christopher/Documents/Studium/Master Thesis/Orientation/sample.csv")
# summary(deter)
dates = strsplit(as.vector(deter$dates), ';')
dates = unlist(dates)
dates = as.Date(dates, format="%Y-%m-%d")
evi = strsplit(as.vector(deter$evi), ';')
evi = as.numeric(unlist(evi))

zoots = zoo(evi, dates, frequency=24)
# check whether the time series is regular. If regular use ts(), if not zoo()
isRegular = is.regular(zoots, strict = FALSE)
isRegular
```

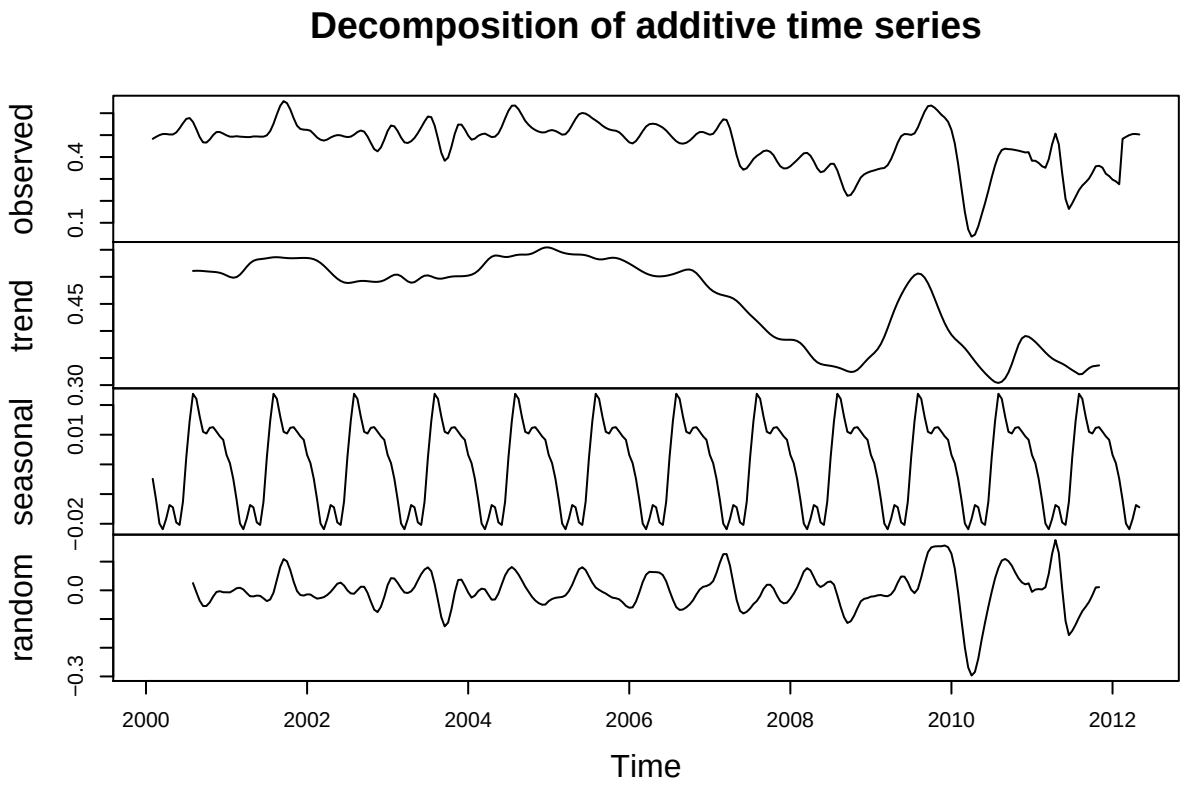
```
## [1] TRUE
```

```r
plot(zoots, typ = "l")
```


```r
# dont use ts for not strict regular / not equispaced time series
tsts = ts(evi, start = c(2000, 3), end = c(2012, 9), frequency = 24)
plot(tsts)
```

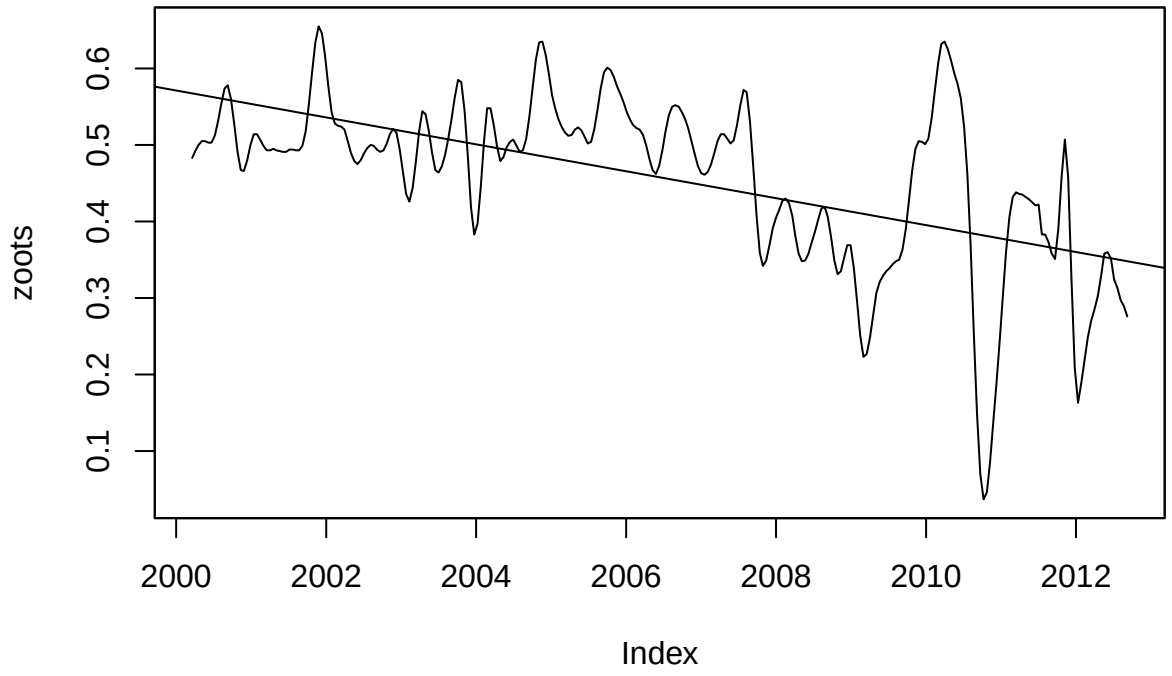



```r
tsts_decom = decompose(tsts, type = "additive")
plot(tsts_decom)
```

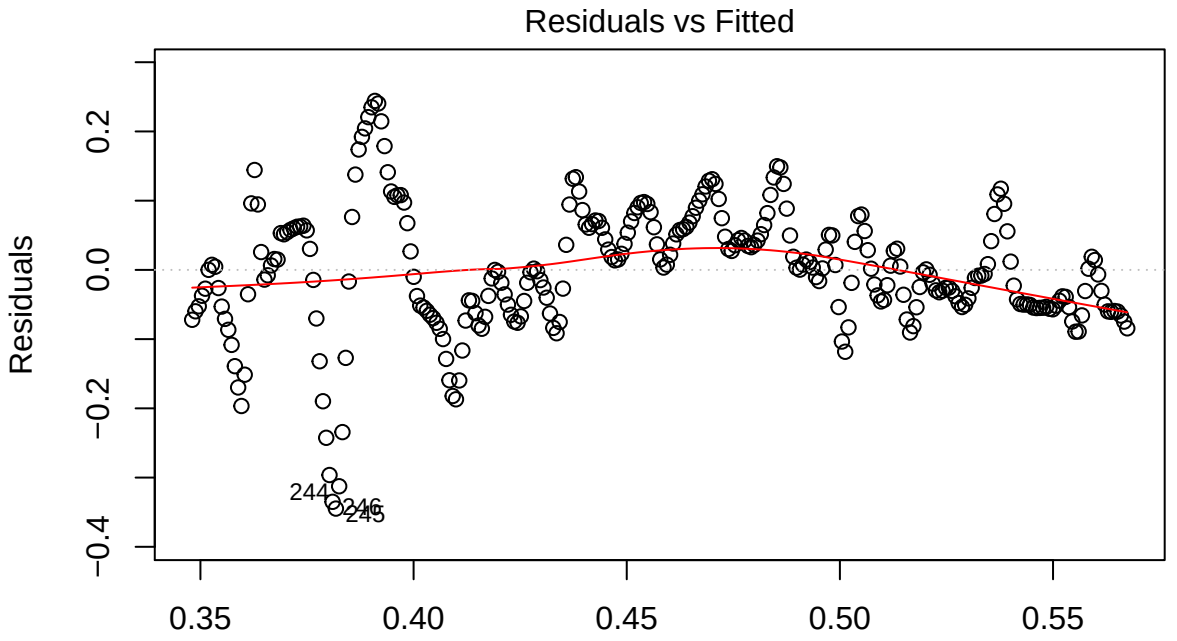


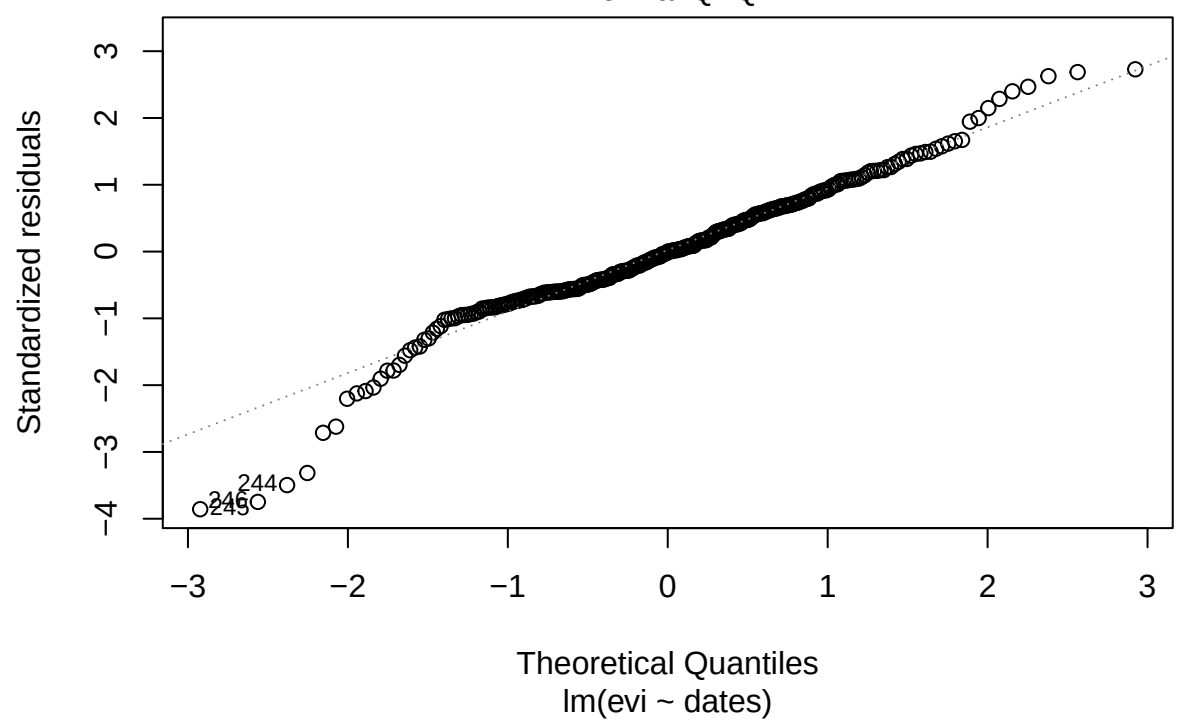
Do the linear regression on the time series.

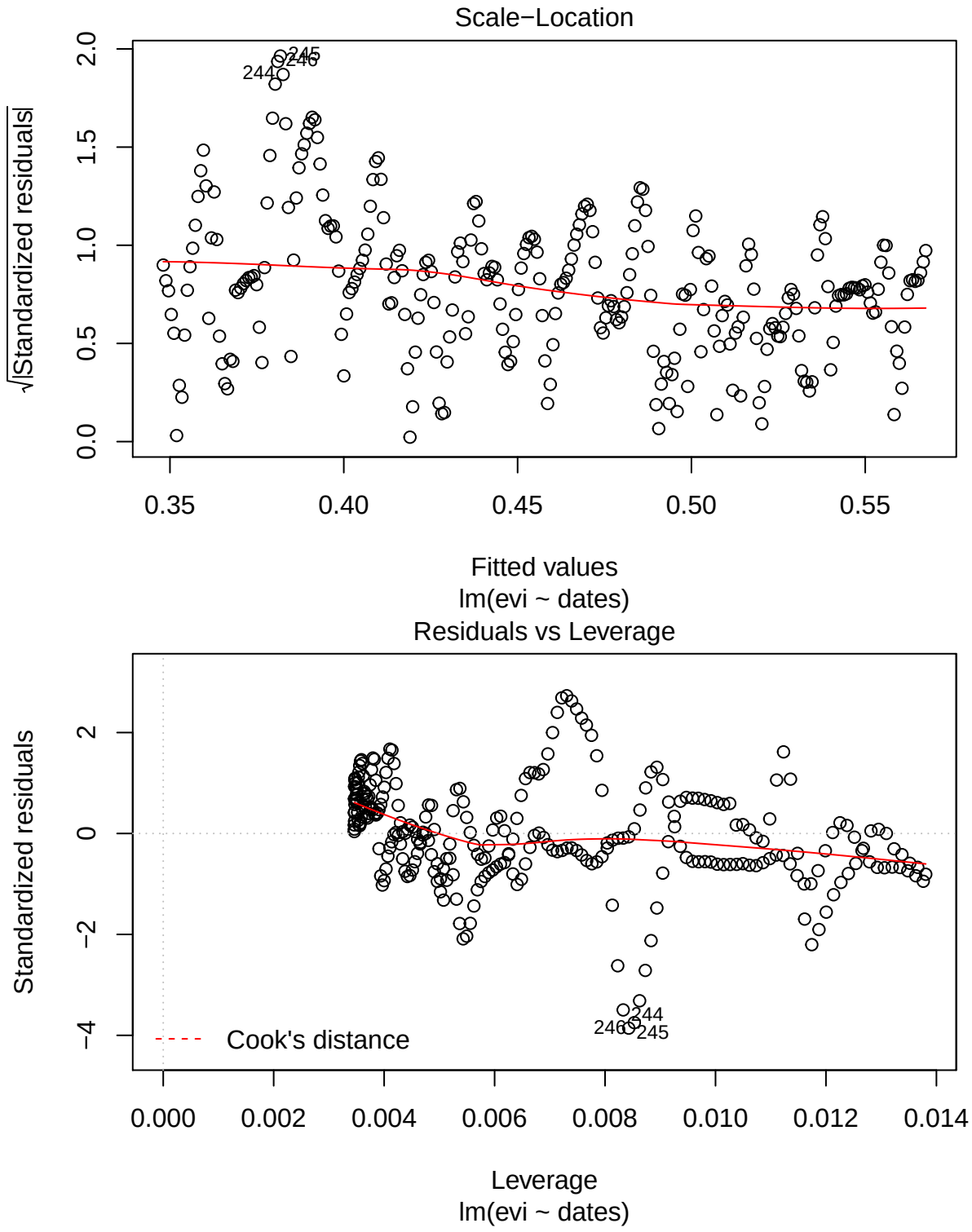
```
lm = lm(evi ~ dates)
plot(zoots)
abline(lm)
```



```
plot(lm)
```


Residuals vs Fitted
Residuals
0.2
0.0
−0.2
−0.4
244
246
245
0.35
0.40
0.45
0.50
0.55
Fitted values
lm(evi ~ dates)


Normal Q−Q
Standardized residuals
3
2
1
0
−1
−2
−3
−4
244
246
245
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(evi ~ dates)


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
245
246
244
0.35
0.40
0.45
0.50
0.55
Fitted values
lm(evi ~ dates)
Residuals vs Leverage
Standardized residuals
−4
−2
0
2
Cook's distance
244
246
245
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
Leverage
lm(evi ~ dates)

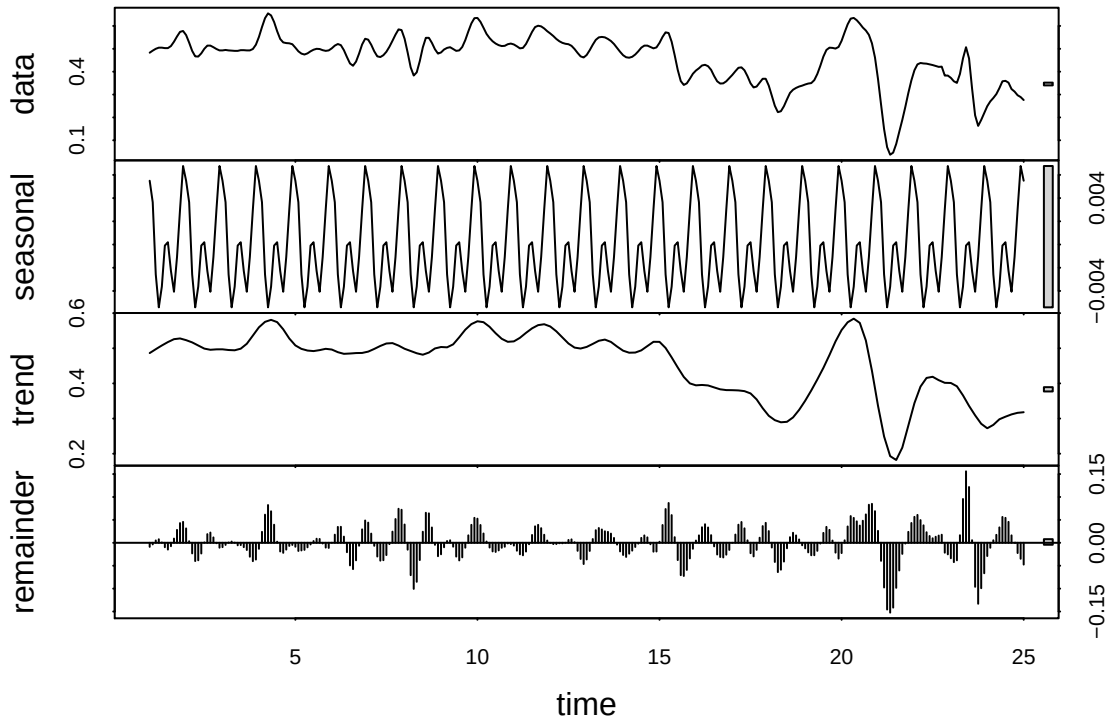
Decompose the time series into seasonality, trend and remainder with stl().

```
na.new <- function(x) ts(na.exclude(x), frequency = 12)
m<-na.exclude(zoots)
m1<-time(m)
m1-m1[-1]
```

```
## Warning: longer object length is not a multiple of shorter object length
```

```
## Time differences in days
##   [1]  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16
##  [15]  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16
##  [29]  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16
##  [43]  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15
##  [57]  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16
##  [71]  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16
##  [85]  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16
##  [99]  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16
## [113]  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15
## [127]  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16
## [141]  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16
## [155]  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16
## [169]  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16
## [183]  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15
## [197]  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16
## [211]  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16
## [225]  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16
## [239]  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16
## [253]  -16  -16  -16  -15  -16  -16  -16  -16  -15  -16  -16  -16  -16  -16
## [267]  -16  -16  -16  -16  -16  -16  -16  -16  -16  -16  -16  -16  -16  -16
## [281]  -16  -16  -16  -16  -16  -16  -16  -16 4540
```

```
zoots_decom = stl(as.ts(zoots), na.action = na.new, s.window = "per")
plot(zoots_decom)
```



Interpret stl() result.

Decomposition procedures are used in time series to describe the trend and seasonal factors in a time series. One of the main objectives for a decomposition is to estimate seasonal effects that can be used to create and present seasonally adjusted values. A seasonally adjusted value removes the seasonal effect from a value so that trends can be seen more clearly. The additive decomposition model is useful when the seasonal variation is relatively constant over time.

The seasonal variation looked to be about the same magnitude across time, so an additive decomposition might be good. There is a major trend change from the year 2009 on. The seasonality is a very regularly repeating pattern.

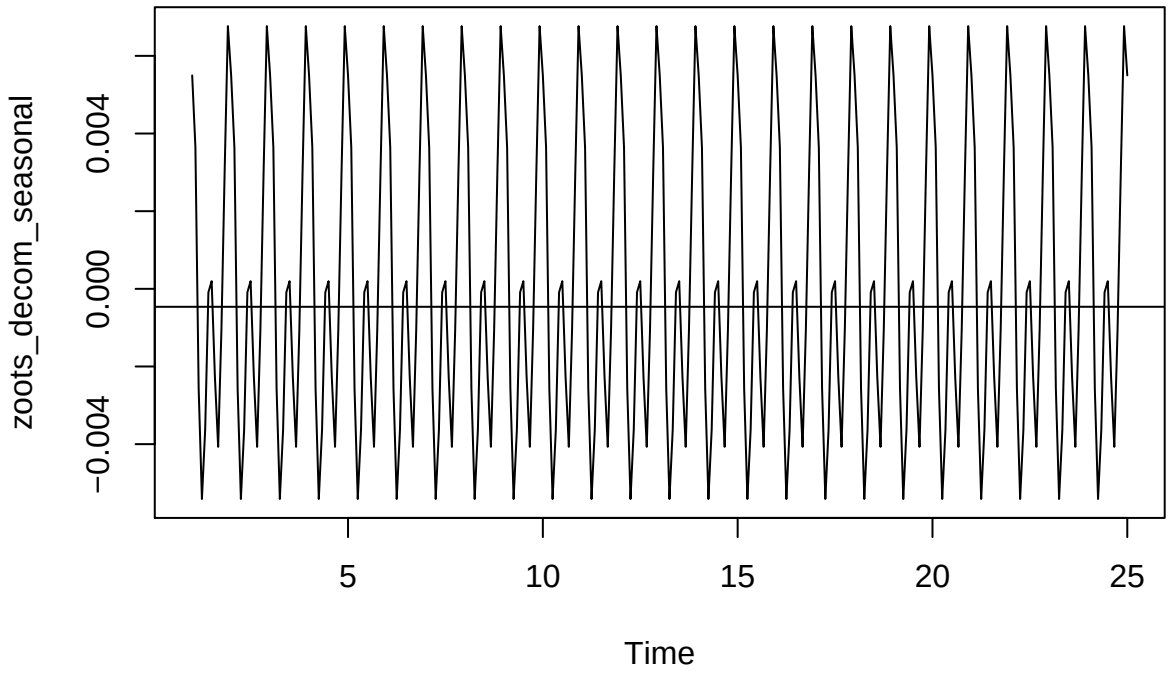
Compute linear regression on seasonal trend and remainder component, respectively.

```
str(zoots_decom)
```

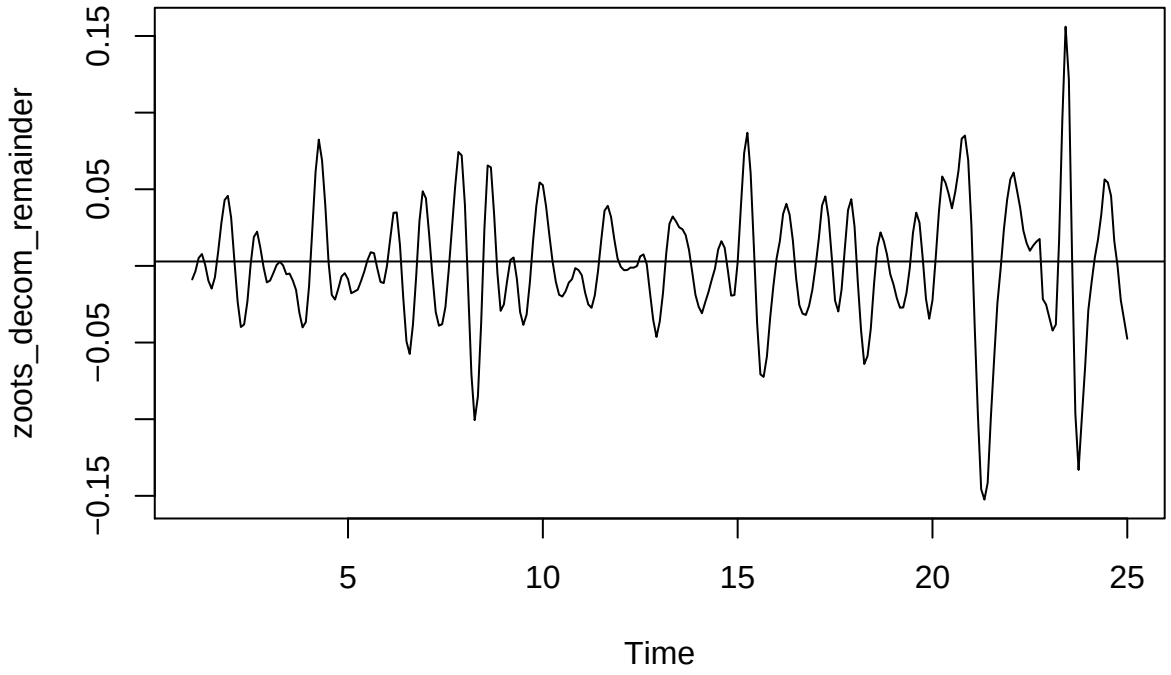
```
## List of 8
##  $ time.series: mts [1:289, 1:3] 0.0055 0.00364 -0.00257 -0.00541 -0.00363 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "seasonal" "trend" "remainder"
##   ..- attr(*, "tsp")= num [1:3] 1 25 12
##   ..- attr(*, "class")= chr [1:3] "mts" "ts" "matrix"
##  $ weights    : num [1:289] 1 1 1 1 1 1 1 1 1 1 ...
##  $ call       : language stl(x = as.ts(zoots), s.window = "per", na.action = na.new)
```

```
##  $ win        : Named num [1:3] 2891 19 13
##   ..- attr(*, "names")= chr [1:3] "s" "t" "l"
##  $ deg        : Named int [1:3] 0 1 1
##   ..- attr(*, "names")= chr [1:3] "s" "t" "l"
##  $ jump       : Named num [1:3] 290 2 2
##   ..- attr(*, "names")= chr [1:3] "s" "t" "l"
##  $ inner      : int 2
##  $ outer      : int 0
##  - attr(*, "class")= chr "stl"
```

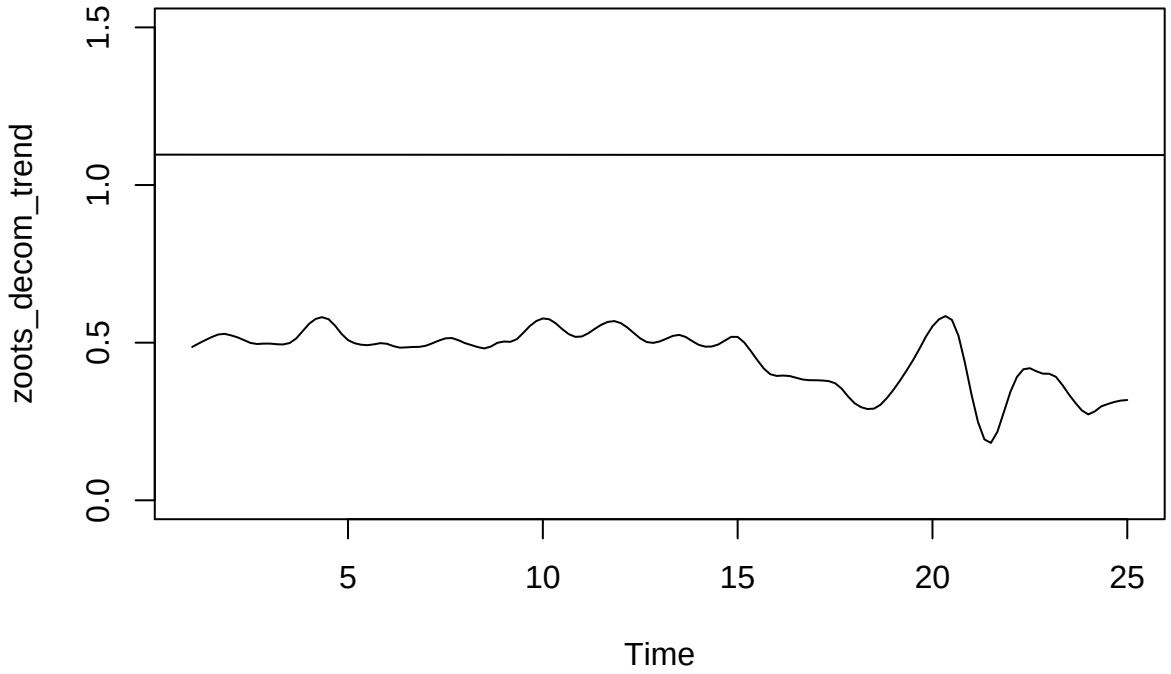
```r
zoots_decom_seasonal = zoots_decom$time.series[, "seasonal"]
plot(zoots_decom_seasonal)
lm_seasonal = lm(zoots_decom_seasonal ~ dates)
abline(lm_seasonal)
```




```r
zoots_decom_remainder = zoots_decom$time.series[, "remainder"]
plot(zoots_decom_remainder)
lm_remainder = lm(zoots_decom_remainder ~ dates)
abline(lm_remainder)
```




```
zoots_decom_trend = zoots_decom$time.series[, "trend"]
plot(zoots_decom_trend,ylim=c(0,1.5))
lm_trend = lm(zoots_decom_trend ~ dates)
abline(lm_trend)
```



Learn about bfast parameters.

Yt univariate time series to be analyzed. This should be an object of class "ts" with a frequency greater than one without NA's.

h minimal segment size between potentially detected breaks in the trend model given as fraction relative to the sample size (i.e. the minimal number of observations in each segment divided by the total length of the timeseries.

season the seasonal model used to fit the seasonal component and detect seasonal breaks (i.e. significant phenological change). There are three options: "dummy", "harmonic", or "none" where "dummy" is the model proposed in the first Remote Sensing of Environment paper and "harmonic" is the model used in the second Remote Sensing of Environment paper (See paper for more details) and where "none" indicates that no seasonal model will be fitted (i.e. St = 0). If there is no seasonal cycle (e.g. frequency of the time series is 1) "none" can be selected to avoid fitting a seasonal model.

max.iter maximum amount of iterations allowed for estimation of breakpoints in seasonal and trend component.

breaks integer specifying the maximal number of breaks to be calculated. By default the maximal number allowed by h is used. hpc | A character specifying the high performance computing support. Default is "none", can be set to "foreach". Install the "foreach" package for hpc support.

Try to perform bfast on the time series.

```
library(bfast)
```

```
## Loading required package: strucchange
## Loading required package: sandwich
## Loading required package: raster
## Loading required package: sp
##
## Attaching package: 'bfast'
##
## The following object is masked _by_ '.GlobalEnv':
##
##     dates
```

```
#tss = ts(rowSums(zoots_decom$time.series), start = c(2000, 3), end = c(2012, 3), frequency=24)
#assign start, end and frequency
#tsp(tss) = tsp(zoots_decom$time.series)

# frequency = 365 / 16 = 23
# f = 23 = annual observations for a 16-days time series
ts = ts(evi, start = c(2000, 6), frequency=23)
length(ts)
```

```
## [1] 289
```

```
# 0.15 * 289 = 43.35 Change cannot be detected within a
# range of h multiplied with the length of the time series
bf = bfast(ts, h=0.15, season="harmonic", max.iter=3)
plot(bf)
```


```
bf
```

```
##
##   TREND BREAKPOINTS
##   Confidence intervals for breakpoints
##   of optimal 4-segment partition:
##
## Call:
## confint.breakpointsfull(object = bp.Vt, het.err = FALSE)
##
## Breakpoints at observation number:
##   2.5 % breakpoints 97.5 %
## 1    91          106    107
## 2   195          197    198
## 3   239          240    242
##
## Corresponding to breakdates:
##   2.5 %    breakpoints 97.5 %
## 1 2004(4)  2004(19)    2004(20)
## 2 2008(16) 2008(18)    2008(19)
## 3 2010(14) 2010(15)    2010(17)
##
##   SEASONAL BREAKPOINTS
##   Confidence intervals for breakpoints
##   of optimal 2-segment partition:
##
## Call:
## confint.breakpointsfull(object = bp.Wt, het.err = FALSE)
##
```

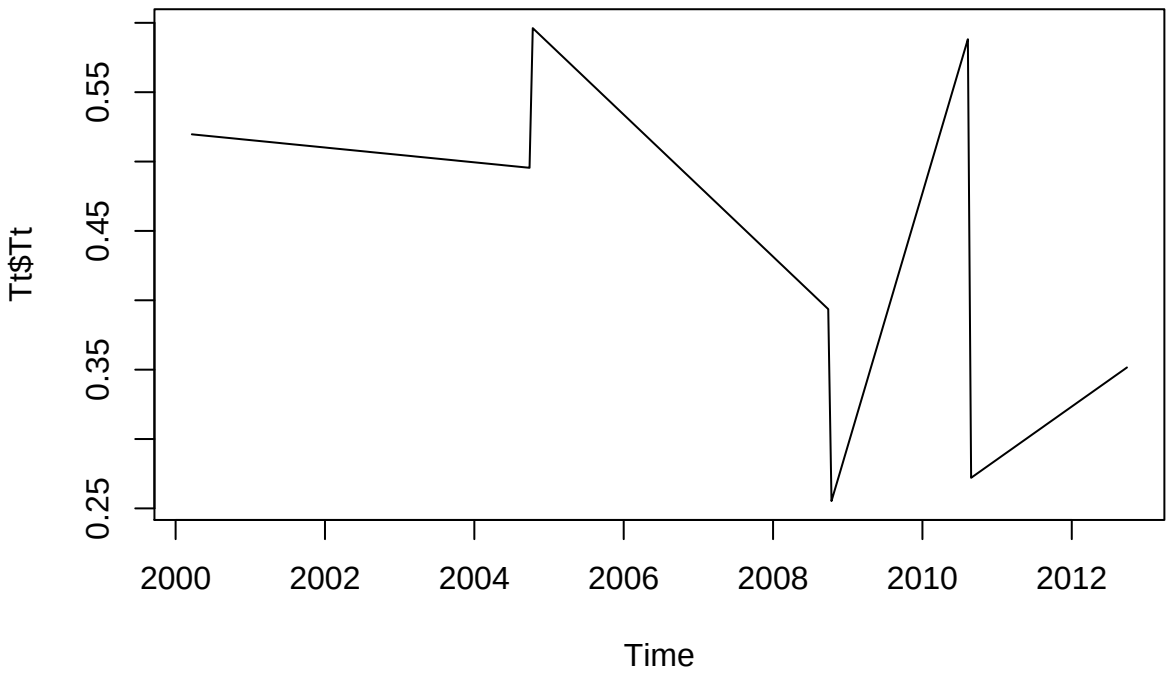
```
## Breakpoints at observation number:
##   2.5 % breakpoints 97.5 %
## 1   207         217    227
##
## Corresponding to breakdates:
##   2.5 %   breakpoints 97.5 %
## 1 2009(5) 2009(15)    2010(2)
```

Interpret the bfast result.

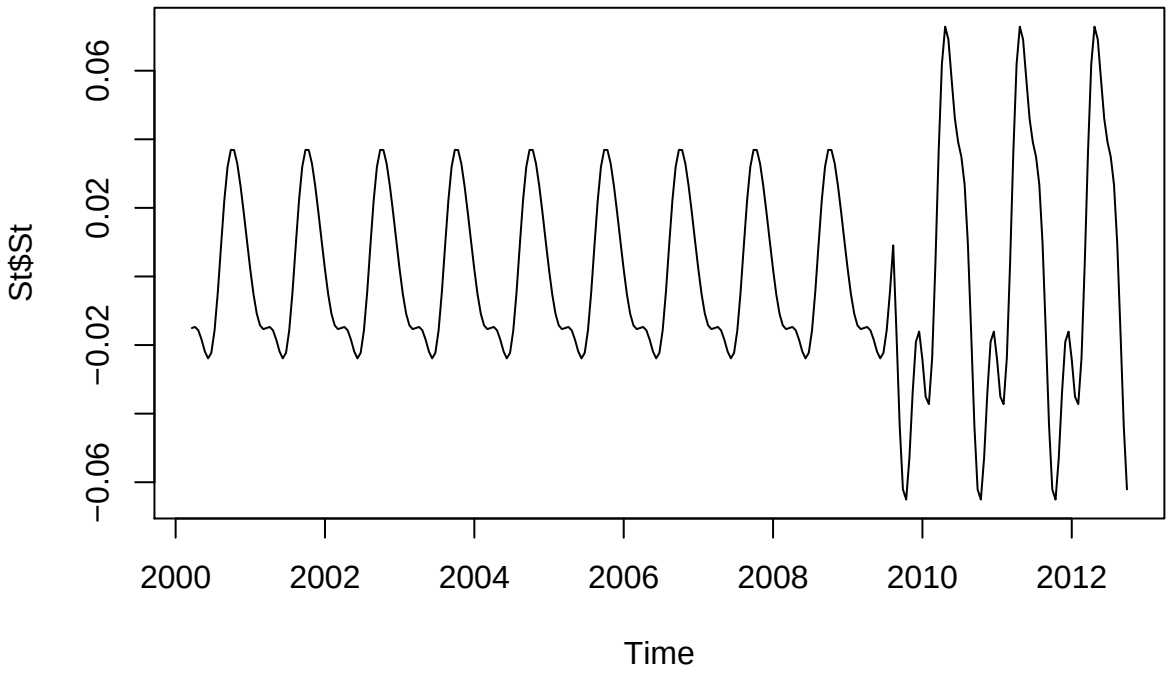
There are two breaks detected in the trend and one in seasonality. The latter one is detected shortly after the second break detection of the trend.

Have a look of all bfast output. Extract bfast output.

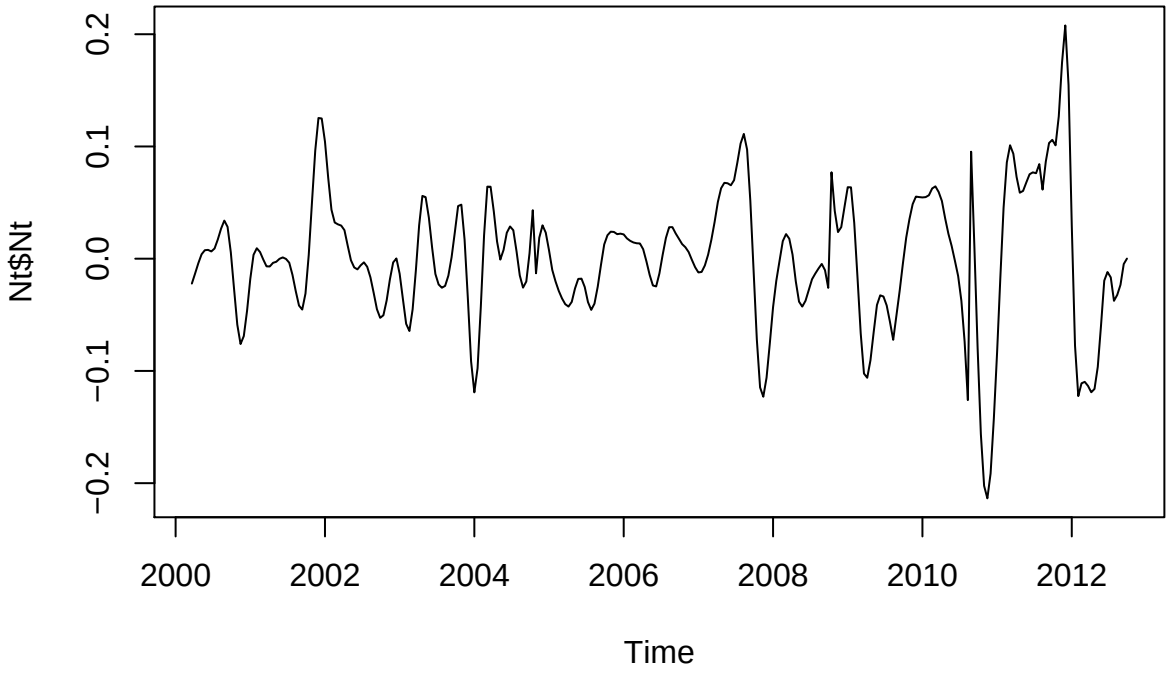
```
#str(bf)

# the trend component
Tt = bf$output[[1]]
plot(Tt$Tt)
```




```
# the fitted seasonal component
St = bf$output[[2]]
plot(St$St)
```




```
# the noise or remainder component
Nt = bf$output[[3]]
plot(Nt$Nt)
```




```
# magnitude of the biggest change detected in the trend component
bf$Magnitude
```

```
## [1] -0.2893
```

```
# logical, TRUE if there are no breakpoints detected
noBrpInTrendModel = bf$nobp[[1]]
noBrpInTrendModel
```

```
## breakpoints
##       FALSE
```

```
# logical, TRUE if there are no breakpoints detected
noBrpInSeasonalModel = bf$nobp[[2]]
noBrpInSeasonalModel
```

```
## breakpoints
##       FALSE
```

```
# timing of the biggest change detected in the trend component
time = bf$Time
dates[time]
```

```
## [1] "2010-07-21"
```

Get time when there is a change in trend or seasonality.

```
bf
```

```
##
##   TREND BREAKPOINTS
##   Confidence intervals for breakpoints
##   of optimal 4-segment partition:
##
## Call:
## confint.breakpointsfull(object = bp.Vt, het.err = FALSE)
##
## Breakpoints at observation number:
##   2.5 % breakpoints 97.5 %
## 1    91         106    107
## 2   195         197    198
## 3   239         240    242
##
## Corresponding to breakdates:
##   2.5 %    breakpoints 97.5 %
## 1 2004(4)  2004(19)    2004(20)
## 2 2008(16) 2008(18)    2008(19)
## 3 2010(14) 2010(15)    2010(17)
##
##   SEASONAL BREAKPOINTS
##   Confidence intervals for breakpoints
##   of optimal 2-segment partition:
##
## Call:
```

```
## confint.breakpointsfull(object = bp.Wt, het.err = FALSE)
##
## Breakpoints at observation number:
##   2.5 % breakpoints 97.5 %
## 1   207         217    227
##
## Corresponding to breakdates:
##   2.5 %   breakpoints 97.5 %
## 1 2009(5) 2009(15)    2010(2)
```

```
# trend breakpoints
trbr1 = dates[106]
trbr1
```

```
## [1] "2004-10-03"
```

```
trbr2 = dates[197]
trbr2
```

```
## [1] "2008-09-09"
```

```
trbr3 = dates[240]
trbr3
```

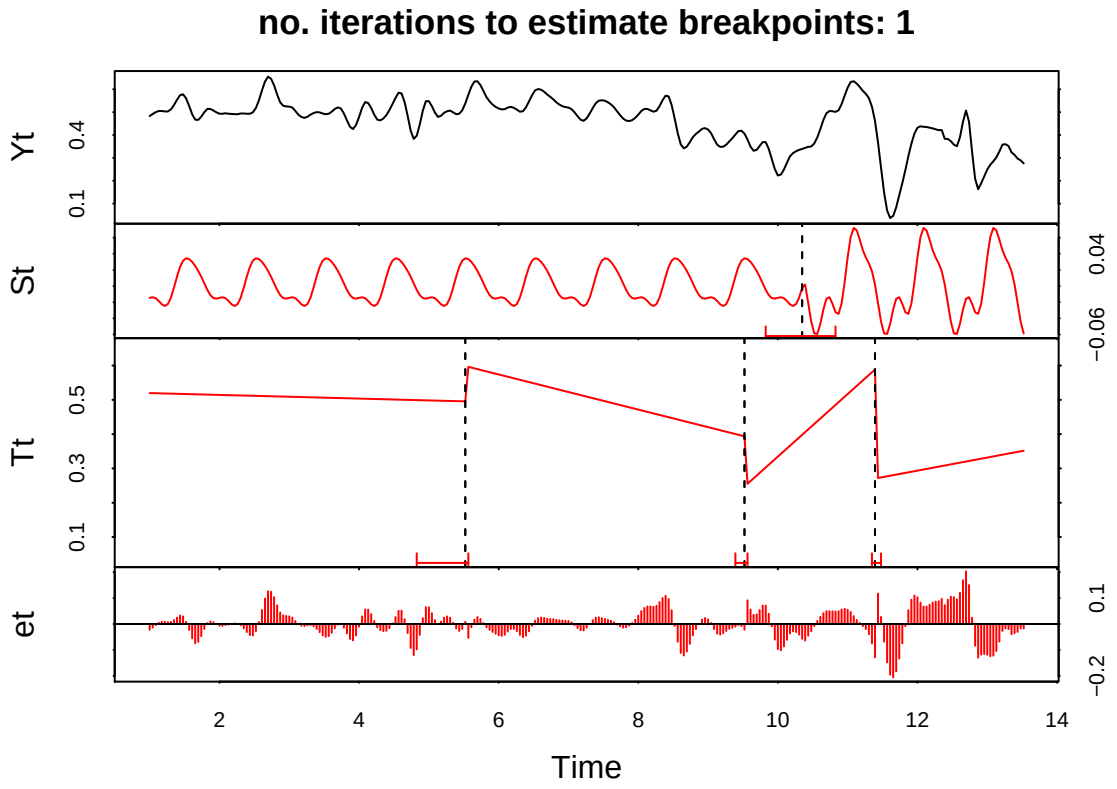
```
## [1] "2010-07-21"
```

```
# seasonal breakpoints
trse = dates[217]
trse
```

```
## [1] "2009-07-22"
```

Perform bfastmonitoring, try to start the monitoring period at different time.

```
ts2 = ts(evi, frequency=23)
bf2 = bfast(ts2, h=0.15, season="harmonic", max.iter=1)
plot(bf2)
```

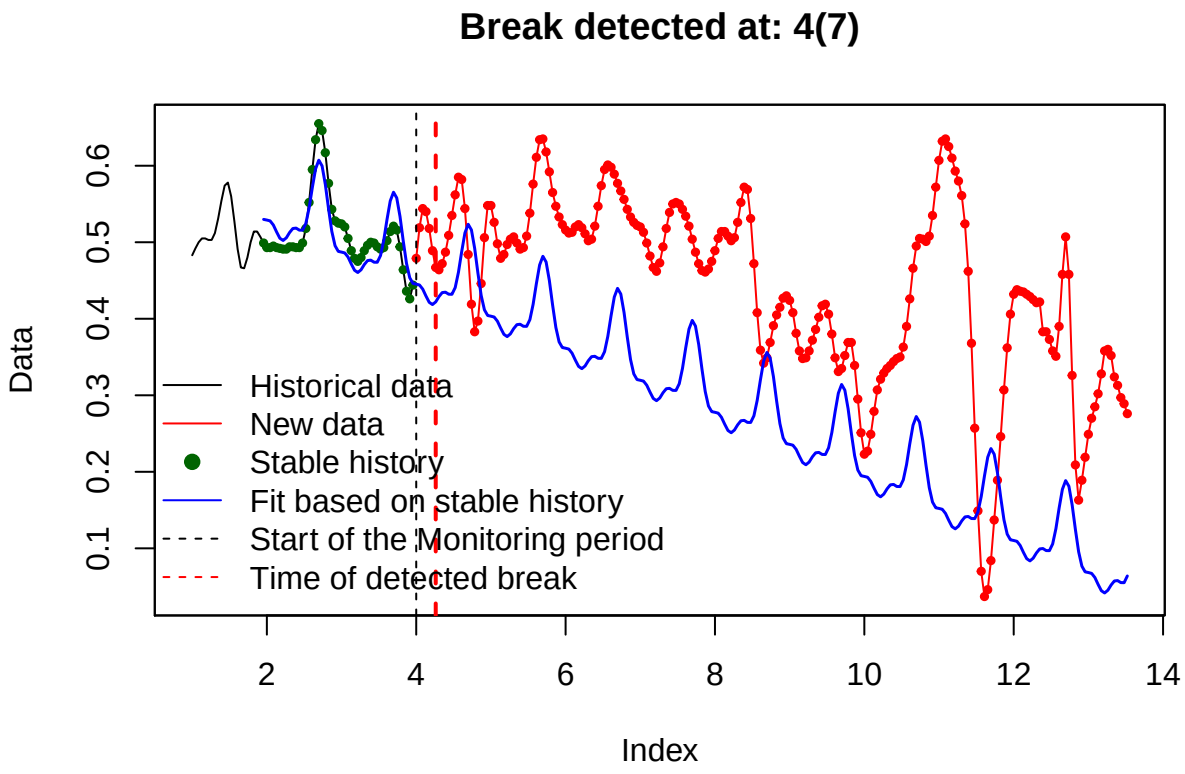



```
bf2
```

```
##
##   TREND BREAKPOINTS
##   Confidence intervals for breakpoints
##   of optimal 4-segment partition:
##
## Call:
## confint.breakpointsfull(object = bp.Vt, het.err = FALSE)
##
## Breakpoints at observation number:
##   2.5 % breakpoints 97.5 %
## 1    89          105    106
## 2   194          197    198
## 3   239          240    242
##
## Corresponding to breakdates:
##   2.5 % breakpoints 97.5 %
## 1 4(20) 5(13)       5(14)
## 2 9(10) 9(13)       9(14)
## 3 11(9) 11(10)      11(12)
##
##   SEASONAL BREAKPOINTS
##   Confidence intervals for breakpoints
##   of optimal 2-segment partition:
##
## Call:
## confint.breakpointsfull(object = bp.Wt, het.err = FALSE)
##
```

```
## Breakpoints at observation number:
##   2.5 % breakpoints 97.5 %
## 1   204         216    227
##
## Corresponding to breakdates:
##   2.5 % breakpoints 97.5 %
## 1 9(20) 10(9)       10(20)
```

```
bfmon1 = bfastmonitor(ts2, start = 4)
plot(bfmon1)
```

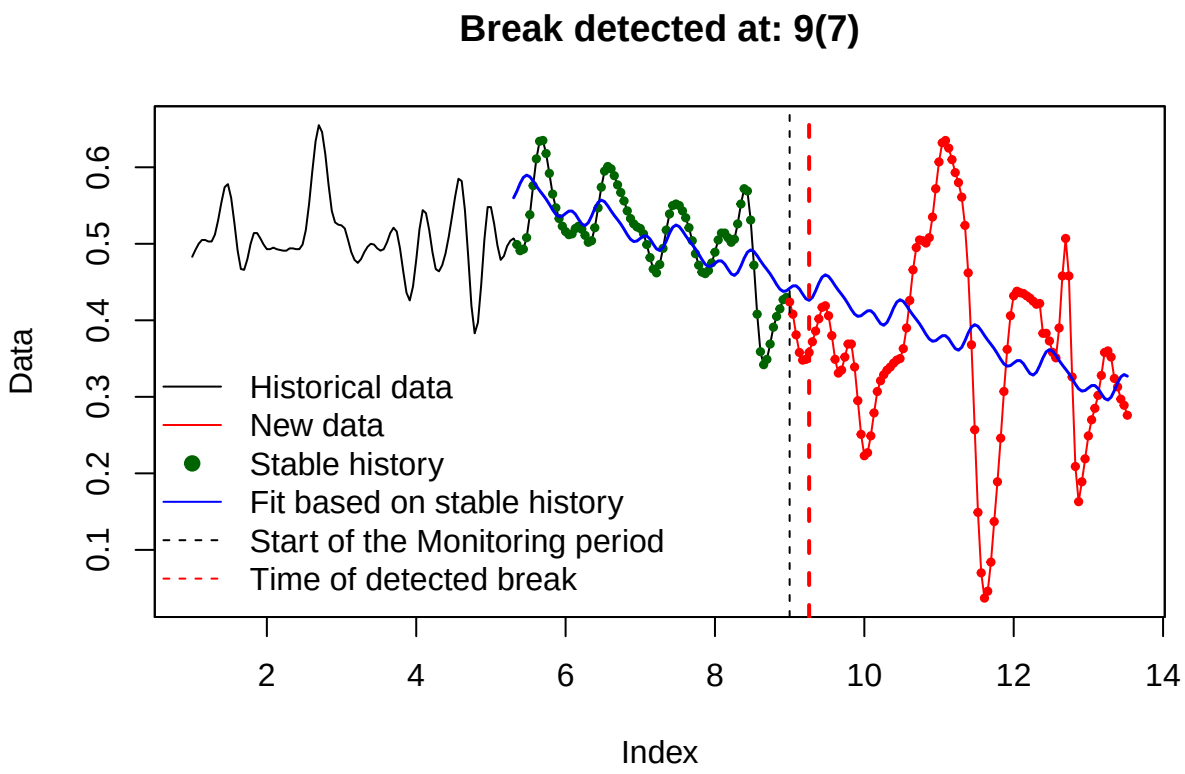



```
bfmon1
```

```
##
## BFAST monitoring
##
## 1. History period
## Stable period selected: 1(23)--3(23)
## Length (in years): 2.000000
## Model fit:
## (Intercept)       trend  harmoncos1  harmoncos2  harmoncos3  harmonsin1
##    0.596204   -0.001821   -0.015840   -0.018196    0.010515   -0.045778
##  harmonsin2  harmonsin3
##    0.011300    0.010004
## R-squared: 0.655080
##
##
## 2. Monitoring period
## Monitoring period assessed: 4(1)--13(13)
```

```
## Length (in years): 9.521739
## Break detected at: 4(7)
```

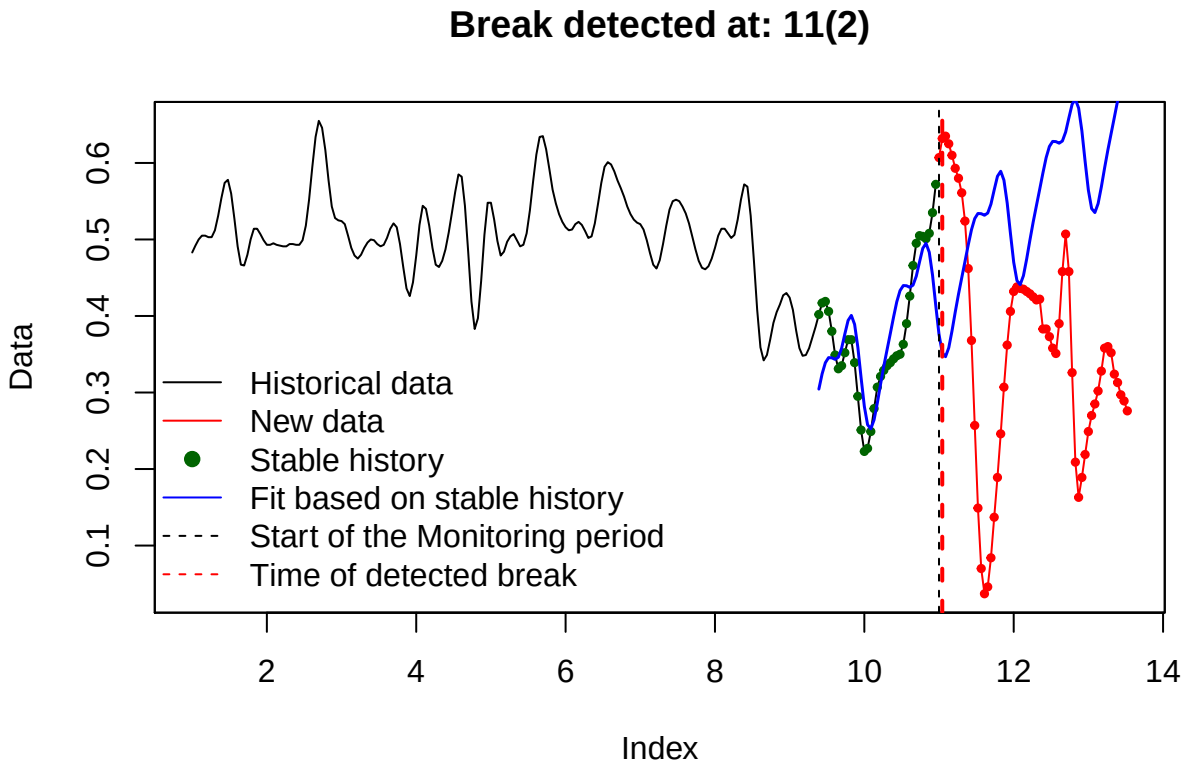
```
bfmon2 = bfastmonitor(ts2, start = 9)
plot(bfmon2)
```




```
bfmon2
```

```
##
## BFAST monitoring
##
## 1. History period
## Stable period selected: 5(8)--8(23)
## Length (in years): 3.652174
## Model fit:
## (Intercept)       trend  harmoncos1  harmoncos2  harmoncos3  harmonsin1
##    0.712559   -0.001416   -0.015127    0.007981   -0.001541   -0.003551
##  harmonsin2  harmonsin3
##    0.004001    0.004817
## R-squared: 0.403894
##
##
## 2. Monitoring period
## Monitoring period assessed: 9(1)--13(13)
## Length (in years): 4.521739
## Break detected at: 9(7)
```

```
bfmon3 = bfastmonitor(ts2, start = 11)
plot(bfmon3)
```




```
bfmon3
```

```
##
## BFAST monitoring
##
## 1. History period
## Stable period selected: 9(10)--10(23)
## Length (in years): 1.565217
## Model fit:
## (Intercept)       trend  harmoncos1  harmoncos2  harmoncos3  harmonsin1
##   -0.506132    0.004095   -0.042279   -0.009450   -0.012005   -0.055207
##  harmonsin2  harmonsin3
##   -0.031030   -0.005308
## R-squared: 0.598868
##
##
## 2. Monitoring period
## Monitoring period assessed: 11(1)--13(13)
## Length (in years): 2.521739
## Break detected at: 11(2)
```

From the table it can bee seen that bfast and bfastmonitor detect breakpoints in the trend and trend-seasonal disturbances roughly at the same time.

bfast trend breakpoints	bfastmonitor trend-seasonal disturbances
5(13)	4(7)
9(13)	9(7)
11(10)	11(2)

Try to understand the differences between bfast monitoring and bfast (theory).

The function *bfast* detects and characterizes trend and seasonal changes within historical time series but the method is not developed to detect disturbances in recently acquired data. The function *bfastmonitor* tries to identify disturbances at the end of time series by comparison with representative, stable, historical data. Bfastmonitor tries to answer the question if new observations t = n, n+1, ... still conform with the the expected behaviour of the historical sample t = 1 ... n. In other words: *Given that a stable-season-trend model was estimated in an observed time period, does it remain stable for new incoming observations?* The crucial assumption of the monitoring approach is that the historical data t = 1 ... n itself is free of disturbances.

Use edit(), fix() to see the function of bfast or bfast monitoring (e.g. fix(bfastmonitoring).

```
edit(bfast)
edit(bfastmonitor)
```