Tutorial: Change points and structural breaks

by Kevin Kotzé

1 Introduction

When working with data in **R** there are many different data objects that may be incorporated within different coding paradigms. For example, you could work with basic vectors and matrixes, or you could make use of variables that are incorporated within one of the many different types of tables. In addition, there are also a number of different types of time series objects, where the individual values for the data are assigned to a specific date.

Broadly speaking there are currently three main types of time series objects that you will frequently encounter:

- `base` variables include basic vectors and matrices, as well as `ts` objects
- `xts` or `zoo` time series variables that are very popular with most financial packages
- `tidyverse` objects that are very popular with the data science fraternity

There are a number of other time series object, such as those in the `tidyquant` suite, which seeks to incorporate features from `xts` and `tidyverse` objects. In what follows, we will largely make use of both `base` and `tidyverse` objects as these are often used in a broad spectrum of projects (i.e. including those are not limited to an investigation that involves time series variables.

The location for the scripts that will be used in this course may be found by following the link one the webpage https://www.economodel.com/time-series-analysis (https://www.economodel.com/time-series-analysis). Please download the appropriate `*.zip` folder. Within each `*.zip` folder, you will find an `*.Rproj` file that sets the working directory and the default environment for the session. In this case, after extracting the files in the `T2-break.zip` folder you should double click on `T2-break.Rproj`, which should open up in **RStudio**. If it doesn't then you will need to right-click on the file before selecting *open-in* and then **RStudio**.

2 Change point tests

Now that the project is open, we can then look at the first script for this session. Please open the file `T2_change.R`. To complete this tutorial, we will need to install a number of packages, where we will be making use of the `changepoint`, `strucchange`, `tidyverse` and `lubridate` packages. These packages are used for: performing change point tests, performing structural change tests, working within the tidyverse environment, and working with dates within the tidyverse environment. In addition, we are also going to collect data that is located in a **GitLab** repository, for which we need the `devtools` package.

To install the packages that are located on **CRAN** we could either click on the `Packages` tab, within **RStudio** before clicking on the `Install` button or alternatively, we could use the following commands:

```
install.packages("devtools")
install.packages("changepoint")
install.packages("strucchange")
install.packages("tidyverse")
install.packages("lubridate")
```

Please note that you only need to install packages once, so when you start a new session, it will not be necessary to re-install them. Since we are also going to apply the change points and structural break tests to current South African economic data, we are going to download all the data that is released by the central bank, which I've stored within a repository, so that everyone is provided with convenient access. The location for this respository is https://gitlab.com/KevinKotze/sarbcurrent (https://gitlab.com/KevinKotze/sarbcurrent). After installing the `devtools` package you will then be able to execute and install the data package from my **GitLab** repository.

```
devtools::install_gitlab("KevinKotze/sarbcurrent")
```

Now that this task is completed, we can then remove all variables from the current environment and close all the figures that may have been generated. This is performed with the following commands:

```
rm(list=ls())  # remove variables
graphics.off() # close figures
```

The next step is to make sure that you can access the routines in the packages that we installed by making use of the `library` command.

```
library(changepoint)
library(sarbcurrent)
library(tidyverse)
library(lubridate)
```

2.1 Change point in mean - simulated data

The first part of this tutorial makes use of simulated data. To ensure that we all get the same results, we set the seed to a predetermined value, which is `123` in our case.

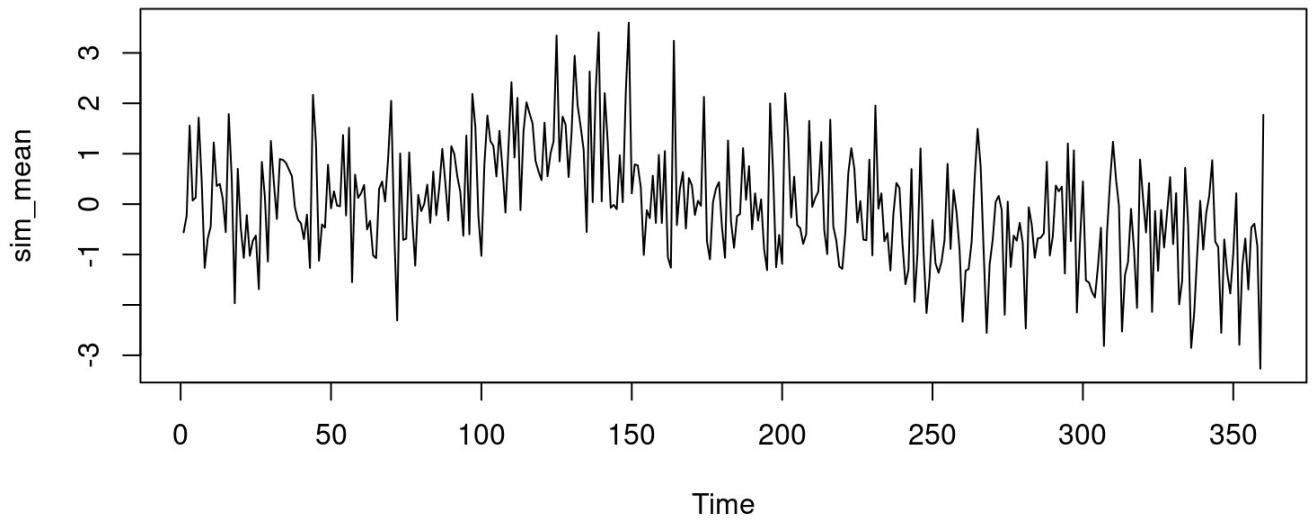
```
set.seed(123)
```

The data will then be drawn from four random normal distributions that have different means and the same variance. The first 100 observations have a mean of zero, the second 50 observations have a mean of 1.5, the third 90 observations have a mean of zero, and the last 120 observations have a mean of -0.8. Each of these sub-samples are then concatenated together with the aid of the `c()` command.

```
sim_mean <- c(rnorm(100, 0, 1),
              rnorm(50, 1.5, 1),
              rnorm(90, 0, 1),
              rnorm(120, -0.8, 1))
```

To plot the time series we can use the `plot.ts`, which is quick and easy to use. There are more attractive ways of producing figures, which we will discuss later in the course, but this is usually one of the most expedient.

```
plot.ts(sim_mean)
```



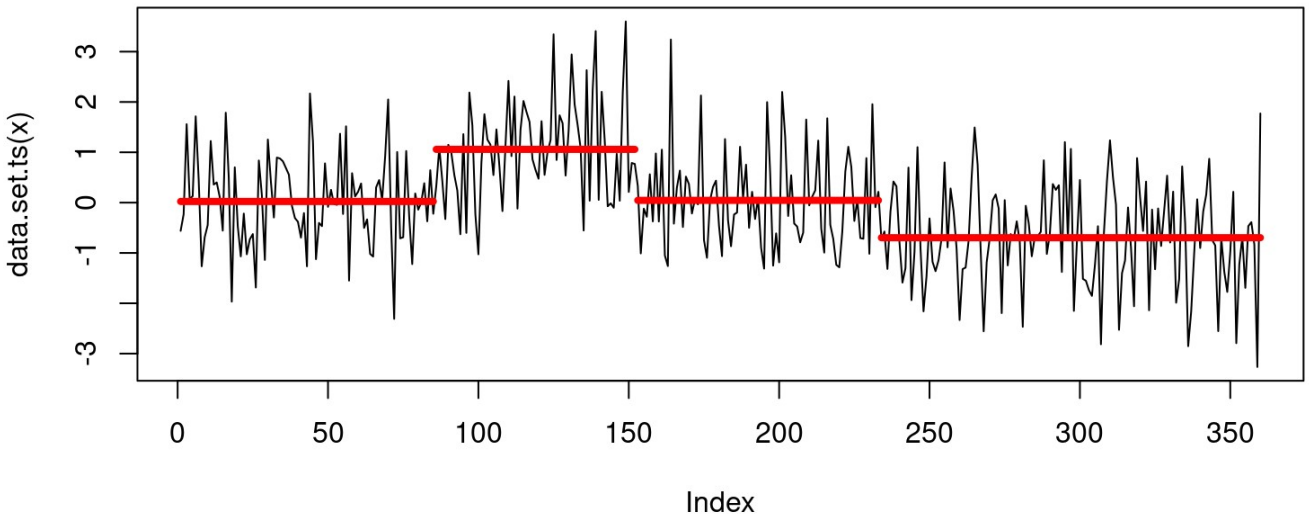
The next few commands will be used to find a potential change point in the mean. We use the `changepoint` package for this task and you will be able to a description of the routines that have been included in this by clicking on the `Packages` tab in **RStudio**. Thereafter, click on the name of the package and you will see a list of the routines. We are going to use the `cpt.mean()` function and if you want to find all the input arguments for this function, then click on the name `cpt.mean()`. You should also find a number of references for this particular routine, as well as an example at the bottom of this screen.

In the first example that we are going to use, we implment the binary segmentation method for a change point test, using the BIC information criteria and a maximum of five breaks. Hence, we would execute the command.

```
m_binseg <- cpt.mean(sim_mean, penalty = "BIC", method = "BinSeg", Q = 5)
```

All the output from the test is assigned to the object, `m_binseg`, which we have created. To plot the results on a graph we can use the `plot` command, as follows:

```
plot(m_binseg, type = "l", xlab = "Index", cpt.width = 4)
```



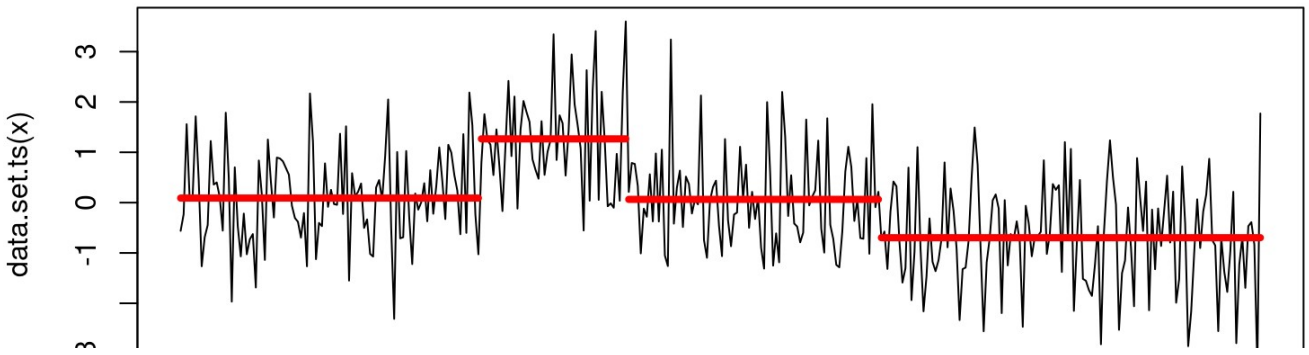
And for details relating to the observation number for the break, we query the `m_binseg` object with the `cpts()` function:

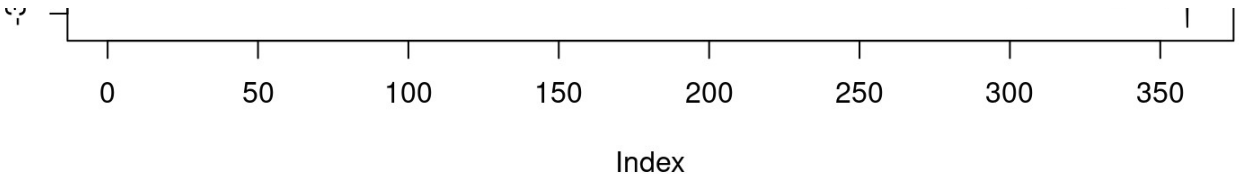
```
cpts(m_binseg)
```

```
## [1]  85 152 233
```

In the following example we make use of the segmented neighbour method, which makes use of all the avialable data in the time series and appears to provide a slightly more accurate estimate of the change points in the mean.

```
m_segneigh <- cpt.mean(sim_mean, penalty = "BIC", method = "SegNeigh", Q = 5)
plot(m_segneigh, type = "l", xlab = "Index", cpt.width = 4)
```



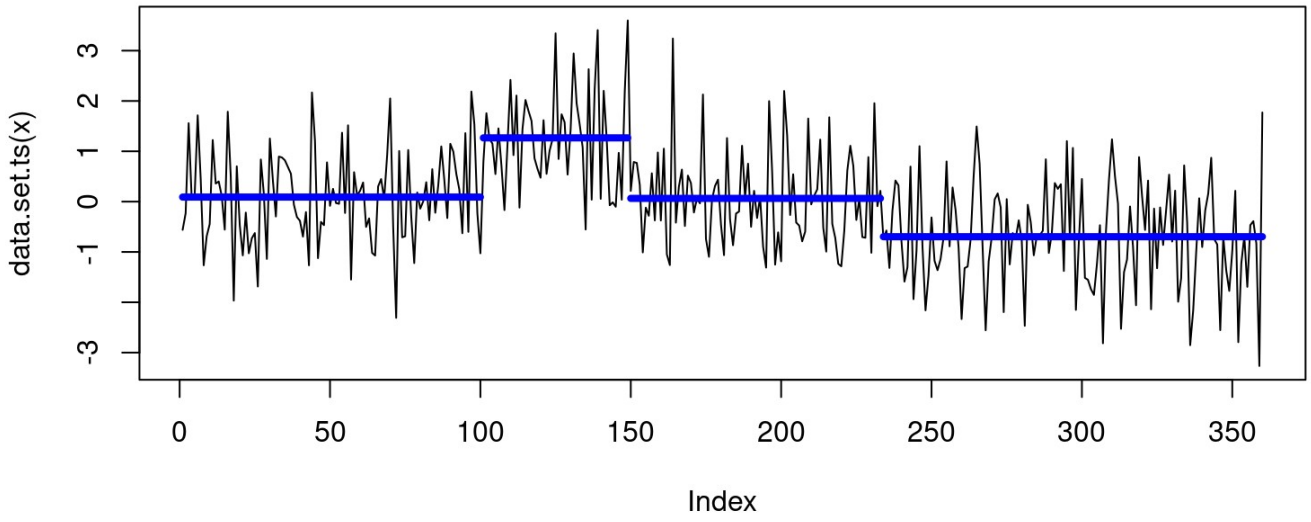



```
cpts(m_segneigh)
```

```
## [1] 100 149 233
```

In addition, we note that the use of the PELT algorithm provides a slightly more expedient result that is equivalent to that the segmented neighbour method.

```
m_pelt <- cpt.mean(sim_mean, penalty = "BIC", method = "PELT")
plot(m_pelt, type = "l", cpt.col = "blue", xlab = "Index", cpt.width = 4)
```

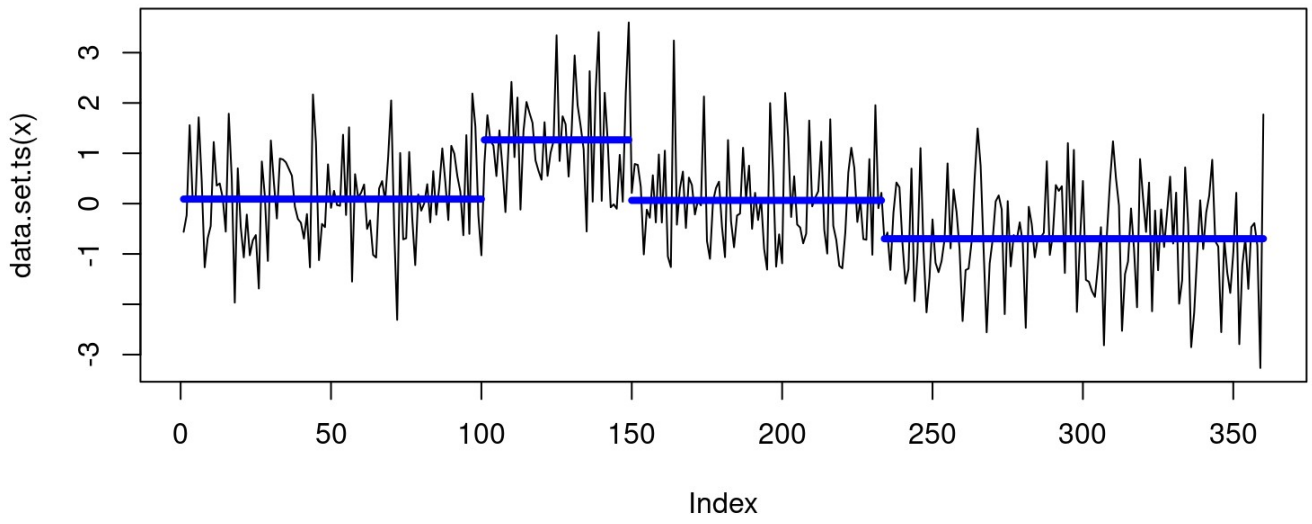



```
cpts(m_pelt)
```

```
## [1] 100 149 233
```

And should we wish, we can also impose a manual penalty with the aid of the following commands:

```
m_pm <- cpt.mean(sim_mean, penalty = "Manual", pen.value = "1.5 * log(n)",
                         method = "PELT")
plot(m_pm, type = "l", cpt.col = "blue", xlab = "Index", cpt.width = 4)
```




```
cpts(m_pm)
```

```
## [1] 100 149 233
```

2.2 Change point in mean - Real Gross Domestic Product

To have a look at the names of the tables that contain the central bank data we can make use of the following command:

```
data(package = 'sarbcurrent')
View(sarb_quarter, title = "sarb_data")
```

The format for this data is consistent with the `tidyverse`, where all the data has combined to form a number of `tibble` objects. When working with such data we can make use of the `pipe` operator, `%>%`, which allows us to write our code in sequential steps. The South African Reserve Bank alpha-numeric code that is used for Real Gross Domestic Product is `KBP6006D`. Therefore, if we want to obtain quarterly GDP data we would make use of the `sarb_quarter` data frame before selecting the columns that relate to the `date` and `KBP6006D`, as follows:

```
sarb_quarter %>%
  select(date, KBP6006D)
```

```
## # A tibble: 392 x 2
##    date       KBP6006D
##    <date>        <dbl>
##  1 1922-04-01       NA
##  2 1922-07-01       NA
##  3 1922-10-01       NA
##  4 1923-01-01       NA
##  5 1923-04-01       NA
##  6 1923-07-01       NA
##  7 1923-10-01       NA
##  8 1924-01-01       NA
##  9 1924-04-01       NA
## 10 1924-07-01       NA
## # … with 382 more rows
```

In addition to `select`, other useful `tidyverse` functions that we will use in this session are: - `mutate()`: creating additional columns - `filter()`: selecting rows acording to the value of a column - `pull()`: extract data out of a tibble - `slice()`: select specific rows based on their position

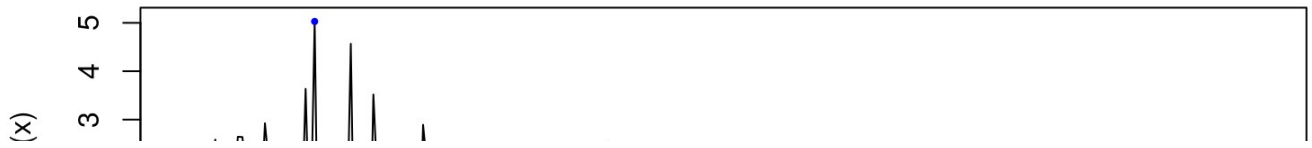
which may be used to create an object `m_gdp` that contains the results for the break point test with the aid of the following commands:

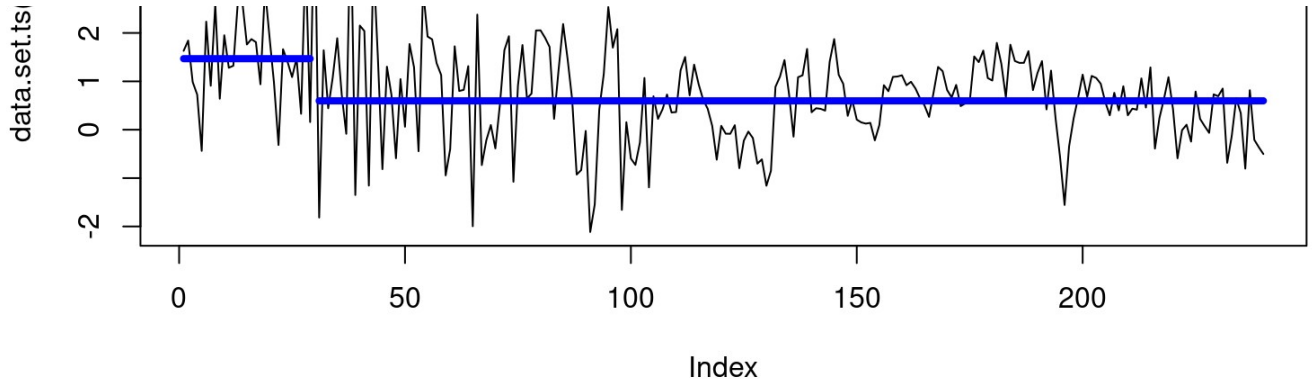
```
m_gdp <- sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1)) %>%
  filter(date > "1960-01-01") %>%
  pull(growth) %>%
  cpt.mean(., penalty = "SIC", method = "PELT")
```

To translate these commands into words, we are looking to make use of the `sarb_quarter` dataset, from which we are selecting `date` and `KBP6006D` columns. Thereafter, we are going to create another column that is called growth, which calculates the growth rate of KBP6006D. Since data for quarterly South African GDP is only available from the first quarter of 1960, we are going to exclude all the rows that relate to dates that are prior to this period. We are then going to pull the growth rate out of the table and will place it in the first placeholder of the `cpt.mean()` function. The result of this calculation will then be assigned to the object `m_gdp`.

We can then plot the result in much the same way as we did before:

```
plot(m_gdp, type = "l", cpt.col = "blue", xlab = "Index", cpt.width = 4)
```





To find the date for the change point we know that the observation number may be found from `cpts(m_gdp)`. Therefore, we can wrangle the data as before and then proceed to make use of `slice()` command, in what follows:

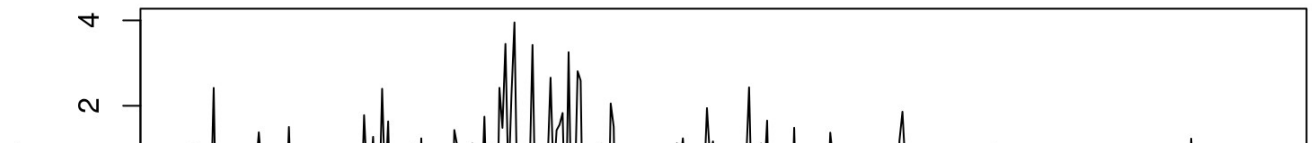
```
sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1)) %>%
  filter(date > "1960-01-01") %>%
  slice(cpts(m_gdp))
```

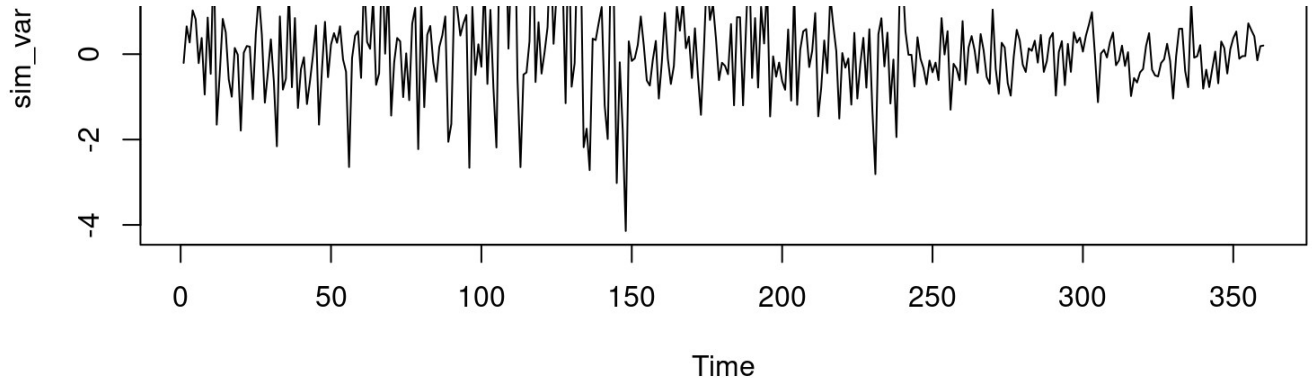
```
## # A tibble: 2 x 3
##   date       KBP6006D growth
##   <date>        <dbl>  <dbl>
## 1 1967-04-01   861388  0.159
## 2 1967-07-01   904702  5.03
```

2.3 Change point in variance - simulated data

To complete a similar exercise for the change in variance we can create a variable `sim_var` that is made up of four parts that have the same mean and different variances. In this case the first 100 observations have a variance of one, the second 50 observations have a variance of two, the third 90 observations have a variance of one, and the last 120 observations have a variance of 0.5.

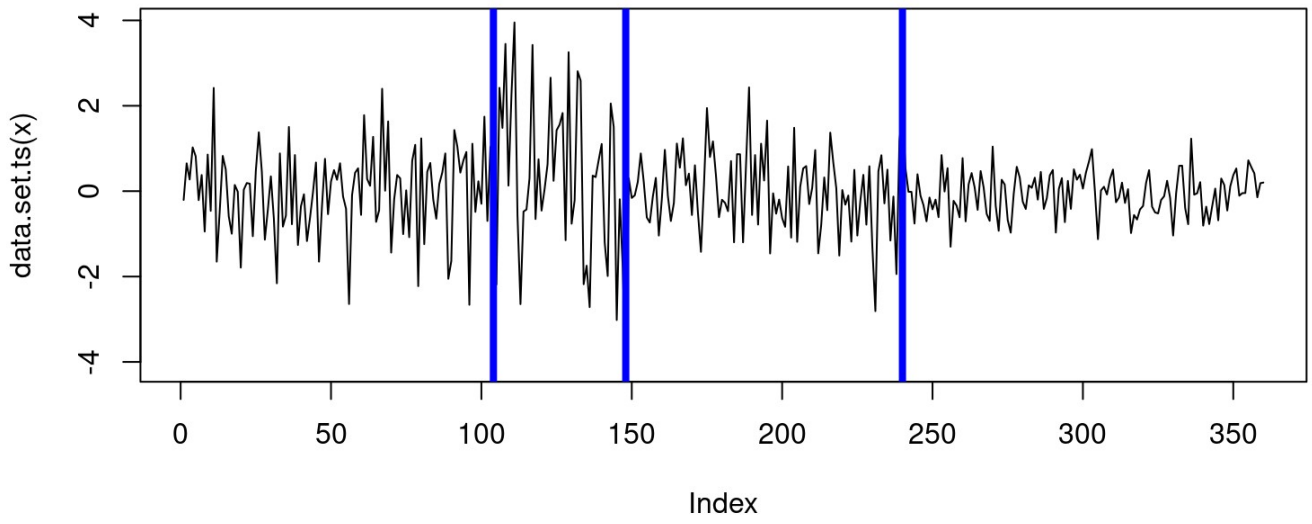
```
sim_var <- c(rnorm(100, 0, 1),
             rnorm(50, 0, 2),
             rnorm(90, 0, 1),
             rnorm(120, 0, 0.5))
plot.ts(sim_var)
```





To perform change point test on variance we make use of the `changepoint` package once again, but on this occassion we make use of the `cpt.var()` function. This syntax could be used to create the object `v.pelt`, as follows:

```
v_pelt <- cpt.var(sim_var, method = "PELT")
plot(v_pelt, type = "l", cpt.col = "blue", xlab = "Index", cpt.width = 4)
```




```
cpts(v_pelt)
```

```
## [1] 104 148 240
```

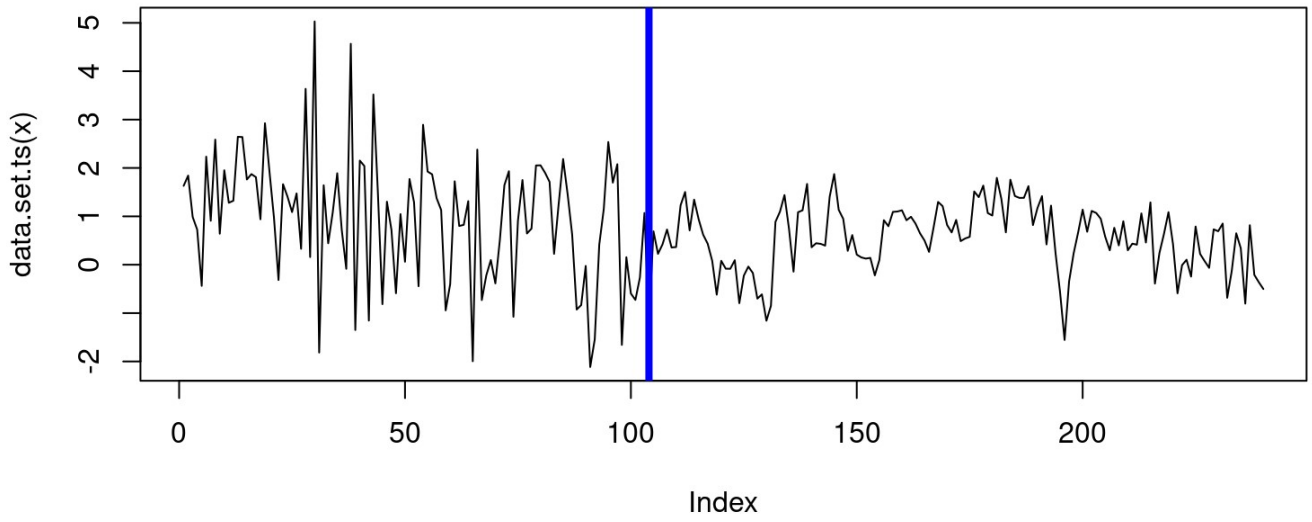
2.4 Change point in variance - Real Gross Domestic Product

When applying this test to South African Real Gross Domestic Product, we would only need to amend the last command, where the results will be assigned to the `v_gdp` object:

```
v_gdp <- sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1)) %>%
  filter(date > "1960-01-01") %>%
  pull(growth) %>%
  cpt.var(., method = "PELT")
```

This object can be used to plot the following result:

```
plot(v_gdp, type = "l", cpt.col = "blue", xlab = "Index", cpt.width = 4)
```



And to find the date for the change points:

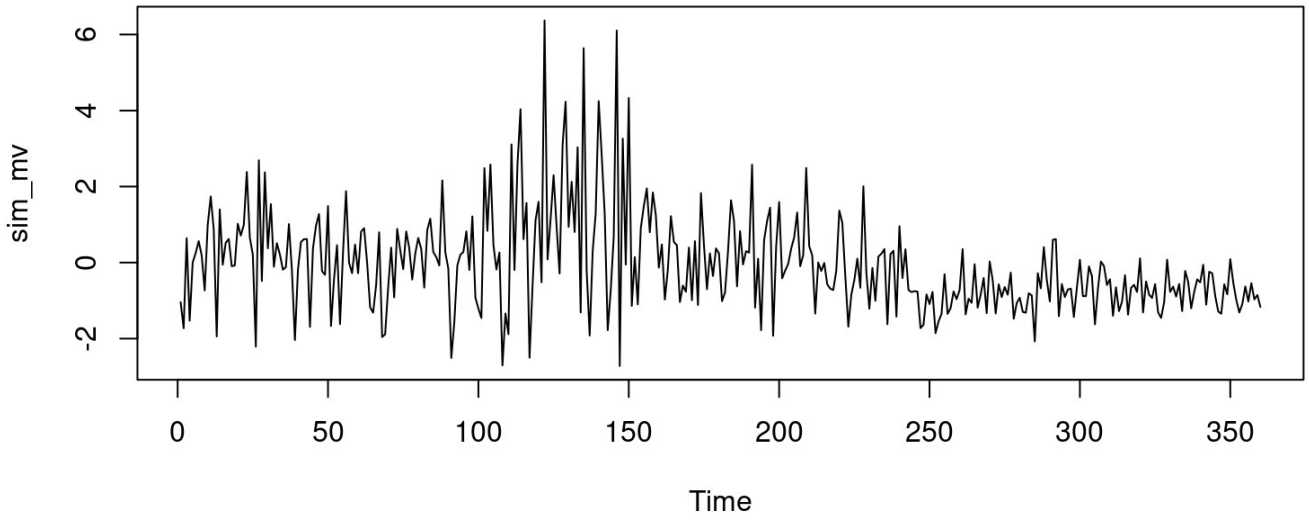
```
sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1)) %>%
  filter(date > "1960-01-01") %>%
  slice(cpts(v_gdp))
```

```
## # A tibble: 1 x 3
##   date       KBP6006D growth
##   <date>        <dbl>  <dbl>
## 1 1986-01-01  1491954  -1.19
```

2.5 Change points in the mean and variance - simulated data

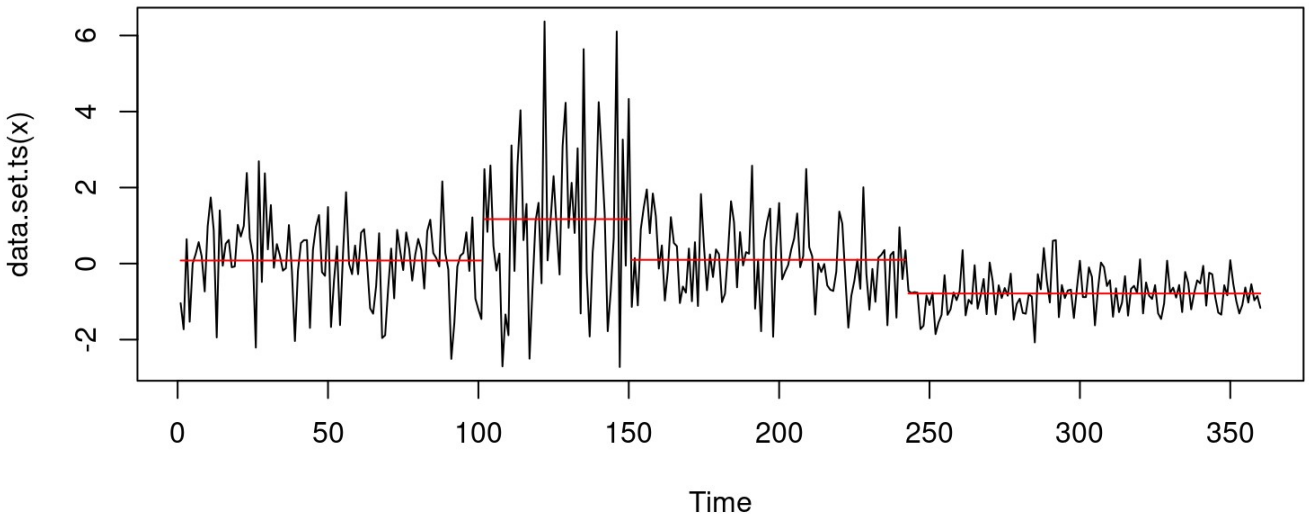
To construct an appropriate variable, which we are going to call `sim_mv`, we make use of the previous assumptions that are applied to both changes in the mean and variance.

```
sim_mv <- c(rnorm(100, 0, 1),
            rnorm(50, 1, 2),
            rnorm(90, 0, 1),
            rnorm(120, -0.8, 0.5))
plot.ts(sim_mv)
```



To then test for a change in the mean and variance we make use of the `cpt.meanvar()` function within the `changepoint` package.

```
mv_pelt <- cpt.meanvar(sim_mv, method = "PELT")
plot(mv_pelt)
```



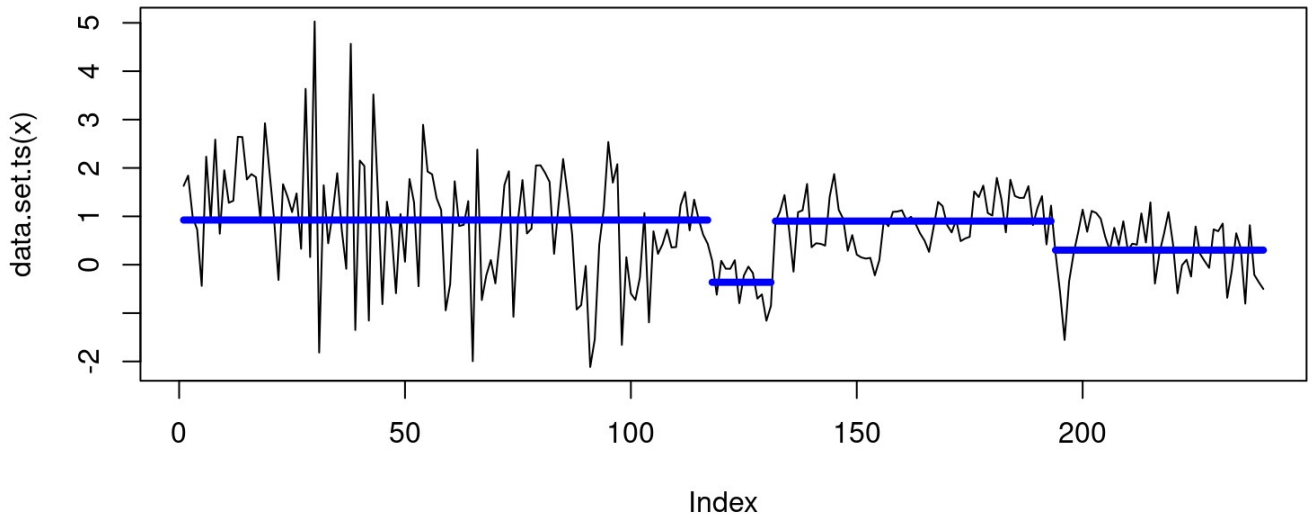
2.6 Change point in mean and variance - Real Gross Domestic Product

When applying this test to South African Real Gross Domestic Product, we would only need to amend the last command once again. In this case, the results will be assigned to the `mv_gdp` object:

```
mv_gdp <- sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1)) %>%
  filter(date > "1960-01-01") %>%
  pull(growth) %>%
  cpt.meanvar(., method = "PELT")
```

To plot the results for the break:

```
plot(mv_gdp, type = "l", cpt.col = "blue", xlab = "Index", cpt.width = 4)
```



And to find the date for the change points:

```
sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1)) %>%
  filter(date > "1960-01-01") %>%
  slice(cpts(mv_gdp))
```

```
## # A tibble: 3 x 3
##   date       KBP6006D growth
##   <date>        <dbl>  <dbl>
## 1 1989-04-01  1640920  0.434
## 2 1992-10-01  1559236 -0.850
## 3 2008-04-01  2717424  1.22
```

3 Structural break tests

The second script that we are going to look at is called `T2_break.R`. After opening this file, we can then remove all variables from the current environment and close all the figures that may have been generated.

```
rm(list=ls())
graphics.off()
```

The next step is to make sure that you can access the routines in the packages that we installed. Note that we are now going to make use of the `strucchange` package and we can access the routines from all the packages that we will use by making use of the `library` command.

```
library(strucchange)
library(sarbcurrent)
library(tidyverse)
library(lubridate)
```

One of the nice features of the `strucchange` package is that it has a detailed user guide. You can find it by going to the packages tab, before clicking on the name of the package. Here you will see a link to `User guides, package vignettes and other documentation`. Thereafter, click on the `*.PDF` option (note that some packages provide such user guides in `HTML` format for these user guides).

3.1 Structural break tests - simulated data

Once again, the first part of this tutorial makes use of simulated datafor which we set the seed to `123`.

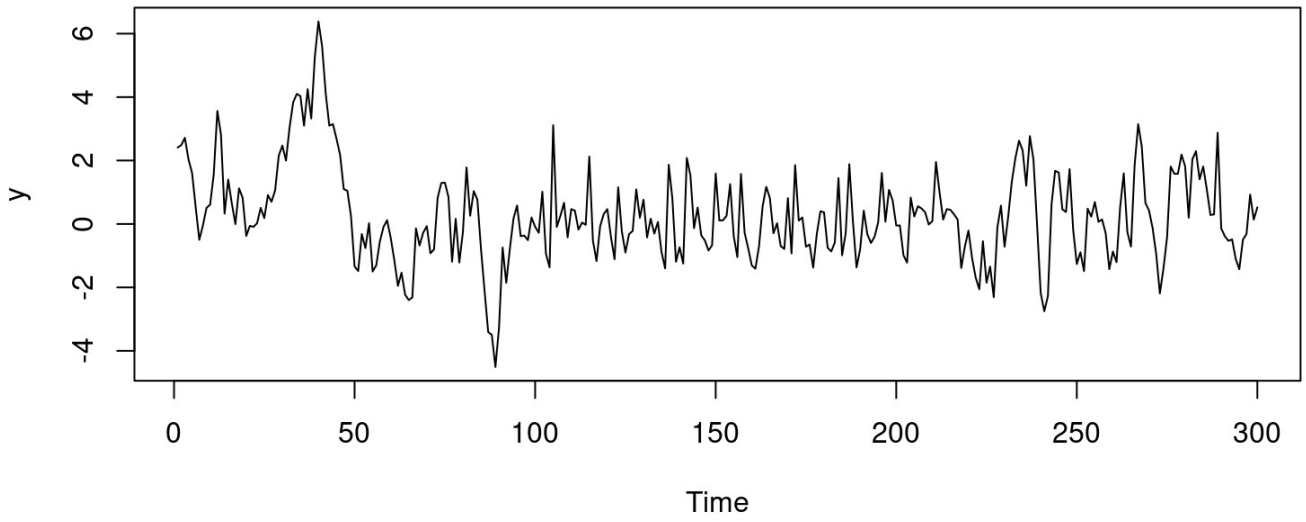
```
set.seed(123)
```

In this case we are going to make use of data that has a certain amount of persistence, where we simulate from autoregressive, moving average and ARMA processes. The first 100 observations are governed by an AR(1) that has a coefficient of 0.9, the second 100 observations are governed by an MA(1) that has a coefficient of 0.1 and the last 100 observations are governed by an ARMA(1,1) that have coefficients of 0.5 and 0.3.

```
x1 <- arima.sim(model = list(ar = 0.9), n = 100)
x2 <- arima.sim(model = list(ma = 0.1), n = 100)
x3 <- arima.sim(model = list(ar = 0.5, ma = 0.3), n = 100)
```

The mean for each of these sub-samples is amended, before they are concatenated together with the aid of the `c()` command to create a variable that is denoted `y` that we plot.

```
y <- c((1 + x1),
       x2,
       (0.5 - x3))
plot.ts(y)
```



The `strucchange` package requires that we specify a regression model, to create residuals that are used to perform the subsequent structural break tests on the residuals. In this case we are going to use an AR(1) regression model so we create database for the left-hand side variable, which is going to be in the first column, while the right-hand side variable (which is the lag of the first column) is placed in the second column. The specific table that we are going to make use of is called a `tibble` and after creating these variables we then drop the rows that have `NA` values.

```
dat <- tibble(ylag0 = y,
              ylag1 = lag(y)
              ) %>%
  drop_na()
```

To generate the Quandt Likelihood Ratio (QLR) statistic we use the `Fstats()` command from the `strucchange` package. Note that the syntax, `ylag0 ~ ylag1` is used in **R** to signify a regression model that has the left-hand side variable `ylag0` , while the regressor is `ylag1`

```
qlr <- Fstats(ylag0 ~ ylag1, data = dat)
```

To show where the structural break arises we use the `breakpoints()` function:

```
breakpoints(qlr)
```

```
##
##   Optimal 2-segment partition:
##
## Call:
## breakpoints.Fstats(obj = qlr)
##
## Breakpoints at observation number:
## 89
##
## Corresponding to breakdates:
## 0.2943144
```

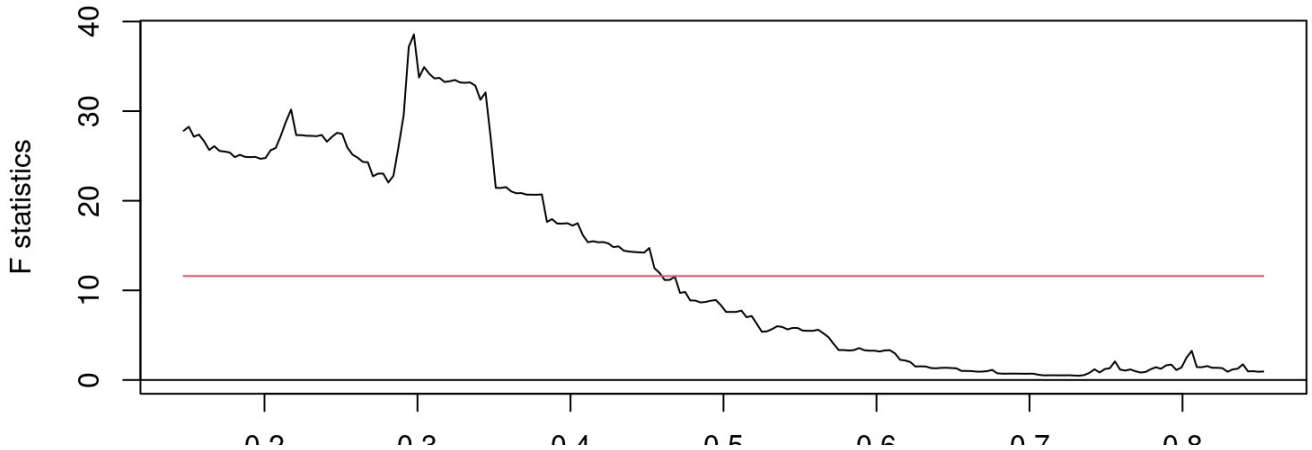
And the significance values for structural break at this point is:

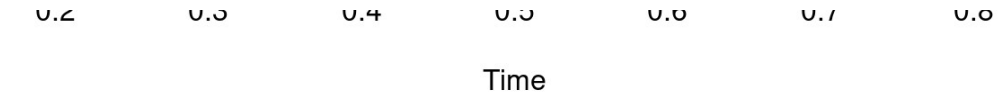
```
sctest(qlr, type = "supF")
```

```
##
##  supF test
##
## data:  qlr
## sup.F = 38.556, p-value = 1.57e-07
```

In addition, we can also present the results with the aid of a figure:

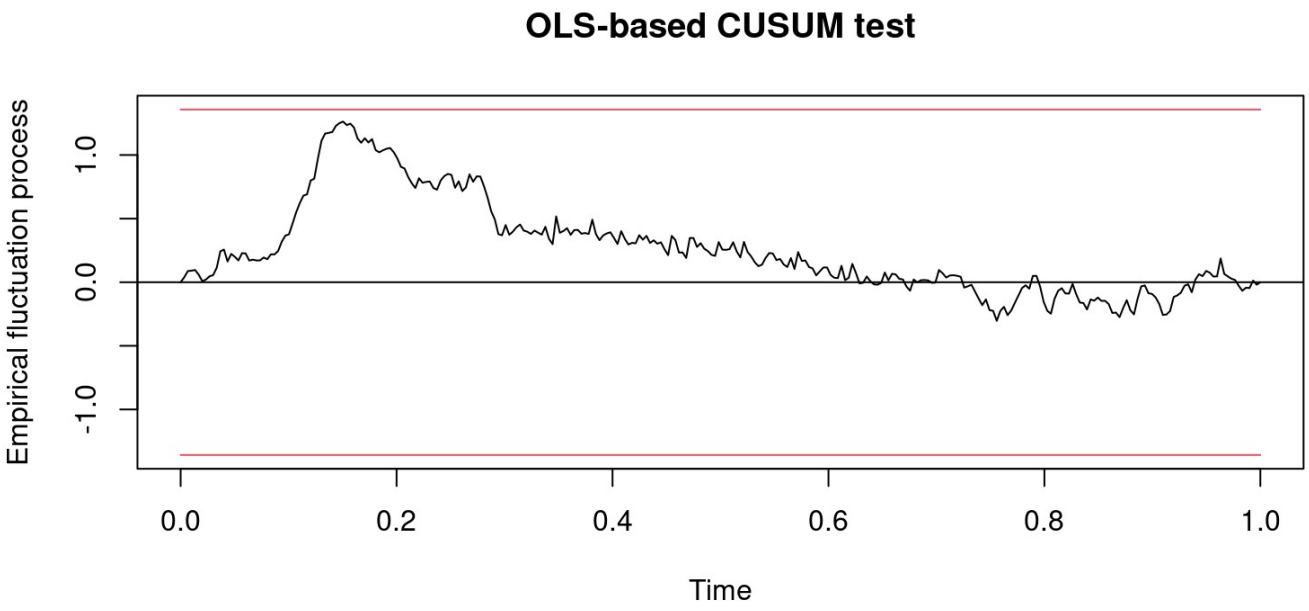
```
plot(qlr)
```





As an alternative to make use of *F*-statistics we can also make make use of the CUSUM statistic, which uses the cumulative sum of the residuals to identify a potential structural break. Such a test may be conducted with the aid of the following commands.

```
cusum <- efp(ylag0 ~ ylag1, type = "OLS-CUSUM", data = dat)
plot(cusum)
```



These results suggest that the data does not contain a structural break, which is incorrect. Note that this is partially due to the fact that we do not have a suitable statistical definition for the term structural break that we can use to test for such an event. We can only test for certain features of a structural break and in this case the cumulative sum of the residuals do not suggest that we have a structural break in this case.

3.2 Structural break tests - Real Gross Domestic Product

In the following example, we are going to make use of the data for South African GDP, which has a certain amount of persistence, so we are going to make use of AR(1) model to obtain the residuals that are hopefully not contain any persistence. Using the commands that we have previously discussed, we create columns for both the growth rate and the lagged growth rate.

```
sa_dat <- sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1),
         grow_lag = lag(growth)) %>%
  drop_na()
```

Thereafter, we generate the QLR statistic using the following commands:

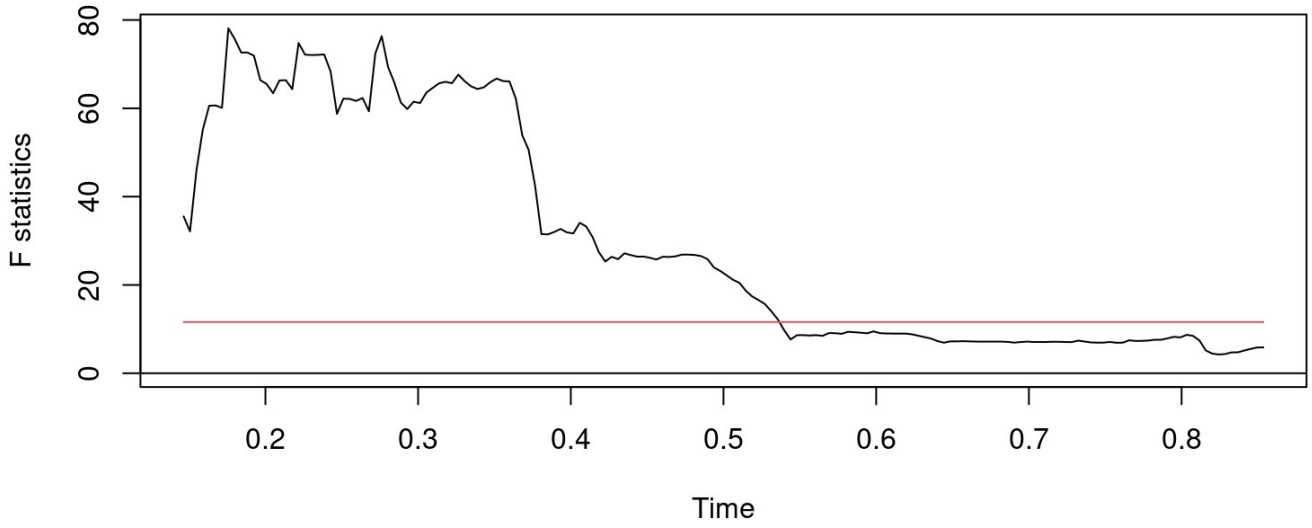
```
sa_qlr <- Fstats(growth ~ grow_lag, data = sa_dat)
breakpoints(sa_qlr)
```

```
##
##   Optimal 2-segment partition:
##
## Call:
## breakpoints.Fstats(obj = sa_qlr)
##
## Breakpoints at observation number:
## 42
##
## Corresponding to breakdates:
## 0.1715481
```

```
sctest(sa_qlr, type = "supF")
```

```
##
##  supF test
##
## data:  sa_qlr
## sup.F = 78.135, p-value = 3.528e-16
```

```
plot(sa_qlr)
```



To find the date that relates to the observation number of the break:

```
sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1),
         grow_lag = lag(growth)) %>%
  drop_na() %>%
  slice(sa_qlr$breakpoint)
```

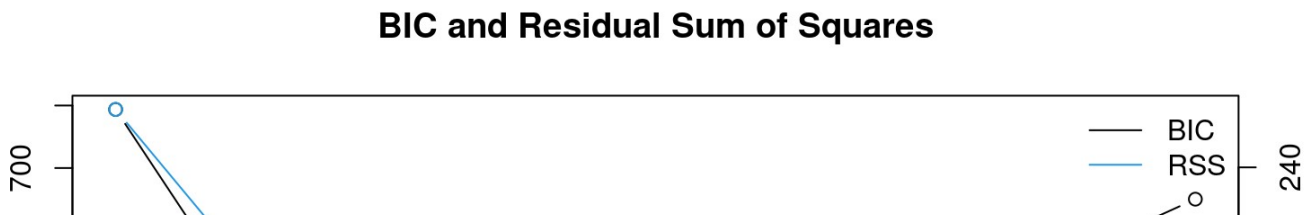
```
## # A tibble: 1 x 4
##   date       KBP6006D growth grow_lag
##   <date>        <dbl>  <dbl>    <dbl>
## 1 1970-10-01  1034117   3.52    -1.15
```

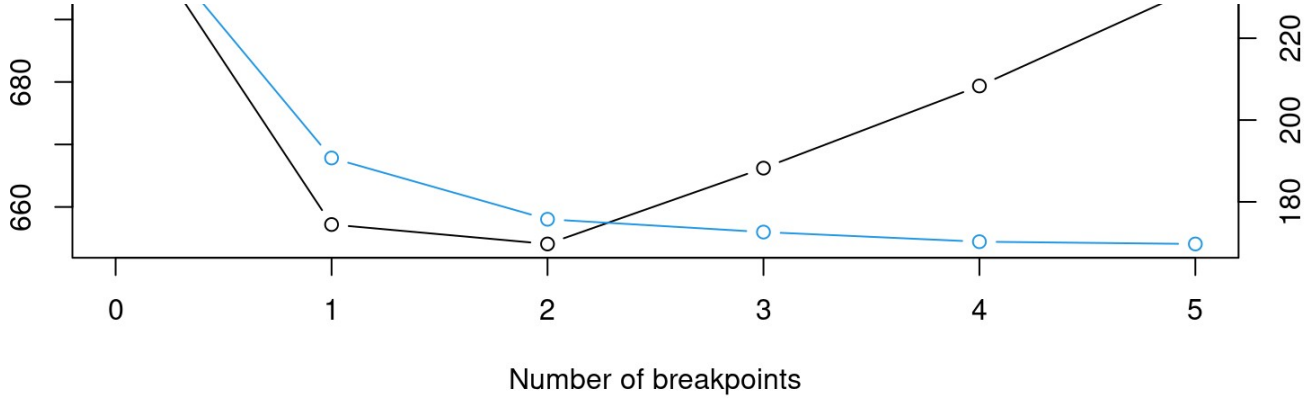
Similarly, we can perform the Bai and Perron (2003) test for multiple structural breaks with the aid of the following commands:

```
sa_bp <- breakpoints(growth ~ grow_lag, data = sa_dat, breaks = 5)
summary(sa_bp)  # where the breakpoints are
```

```
##
##   Optimal (m+1)-segment partition:
##
## Call:
## breakpoints.formula(formula = growth ~ grow_lag, breaks = 5,
##     data = sa_dat)
##
## Breakpoints at observation number:
##
## m = 1   42
## m = 2   42 78
## m = 3   42 78         204
## m = 4   42 78 130     203
## m = 5   42 78 130 166 203
##
## Corresponding to breakdates:
##
## m = 1   0.175732217573222
## m = 2   0.175732217573222 0.326359832635983
## m = 3   0.175732217573222 0.326359832635983
## m = 4   0.175732217573222 0.326359832635983
## m = 5   0.175732217573222 0.326359832635983
##
## m = 1
## m = 2
## m = 3
## m = 4   0.543933054393305
## m = 5   0.543933054393305 0.694560669456067
##
## m = 1
## m = 2
## m = 3   0.853556485355649
## m = 4   0.849372384937238
## m = 5   0.849372384937238
##
## Fit:
##
## m   0     1     2     3     4     5
## RSS 254.1 190.7 175.8 172.6 170.3 169.7
## BIC 709.4 657.2 654.1 666.2 679.4 695.0
```

```
plot(sa_bp, breaks = 5)
```





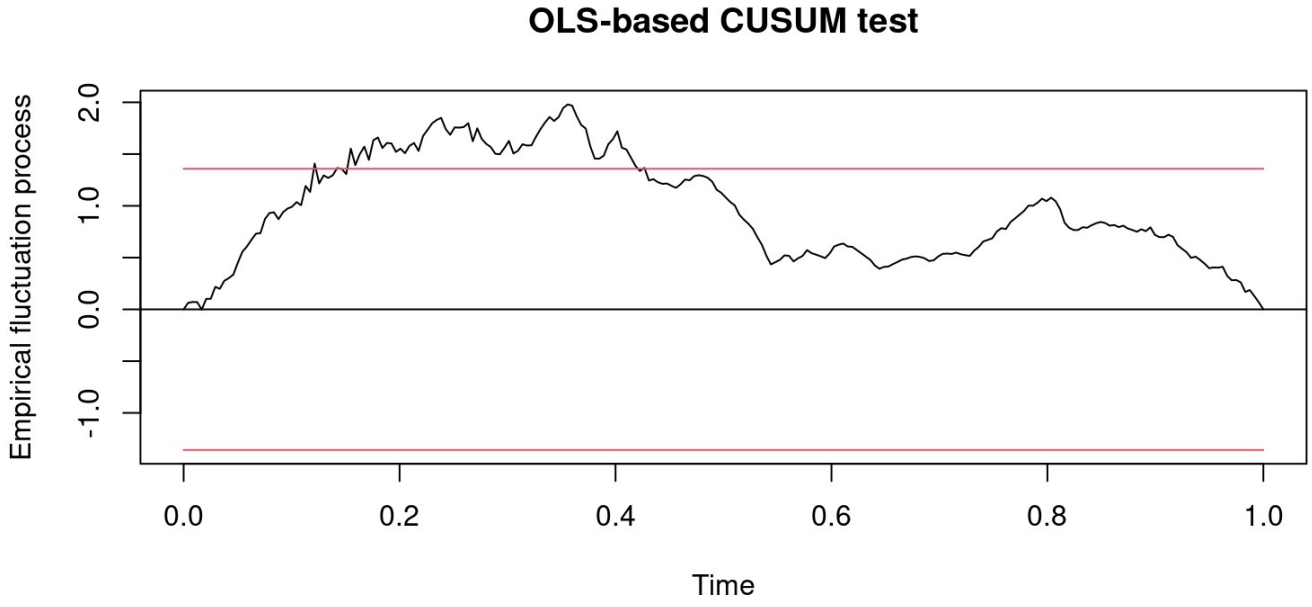
where the date that relates to the observation number of the break is:

```
sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1),
         grow_lag = lag(growth)) %>%
  drop_na() %>%
  slice(sa_bp$breakpoint)
```

```
## # A tibble: 2 x 4
##   date       KBP6006D growth grow_lag
##   <date>        <dbl>  <dbl>    <dbl>
## 1 1970-10-01  1034117   3.52   -1.15
## 2 1979-10-01  1344927   2.05    0.746
```

And the corresponding CUSUM test could be executed with the syntax:

```
sa_cusum <- efp(growth ~ grow_lag, data = sa_dat, type = "OLS-CUSUM")
plot(sa_cusum)
```



4 Conclusion

And that draws this tutorial to end.

Bai, Jushan, and Pierre Perron. 2003. “Computation and Analysis of Multiple Structural Change Models.” *Journal of Applied Econometrics* 18 (1): 1–22.