strucchange: Testing for Structural Change in Linear Regression Relationships

by Achim Zeileis

Introduction

The problem of detecting structural changes arises most often when analyzing time series data with linear regression models, especially in econometrics. Consider the standard linear regression model

$$y_i = x_i^\top \beta_i + u_i \qquad (i = 1, \ldots, n),$$

where at time i, y_i is the observation of the dependent variable, x_i is a vector of regressors, β_i is the k-dimensional vector of regression coefficients and u_i is an iid error term. Tests on structural change are concerned with testing the null hypothesis of "no structural change"

$$H_0: \quad \beta_i = \beta_0 \qquad (i = 1, \ldots, n),$$

i.e., that the regression coefficients remain constant, against the alternative that the coefficient vector varies over time.

These tests can be divided in two classes, which are differently suitable for certain patterns of deviation from the null hypothesis. The first class are the tests from the generalized fluctuation test framework (Kuan and Hornik, 1995) that can detect various types of structural changes. The second class are the tests from the F test framework (Hansen, 1992; Andrews, 1993), which assume that there is one (unknown) breakpoint under the alternative.

In this article we describe the ideas and methods that have been implemented in the package and that reflect the common features of both classes of tests: how the model for a test can be fitted, the results plotted and finally how the significance can be assessed. First we will introduce the tests and then offer an application on some anthropological data.

Generalized fluctuation tests

Fluctuation tests are either based on estimates or on residuals. The idea of the estimates-based tests is, that if there is a structural change in the data the estimate of the regression coefficients on the basis of all data should be substantially different from the estimates on subsamples of the data that do not contain the structural change(s). But these estimates should be rather similar if the true coefficients remain constant over time. Therefore in this case an empirical process can be computed by the differences of these subsample estimates with the overall estimate. The subsamples are either chosen recursively, i.e., starting with the first k observations and including step by step the next observation, or by a window of constant width that "moves" over the whole sample period. The resulting processes should not fluctuate (deviate from zero) too much under the null hypothesis and—as the asymptotic distributions of these processes are well-known—boundaries can be computed, which are only crossed with a certain controlled probability α. If, on the other hand, the empirical process shows large fluctuation and crosses the boundary, there is evidence that the data contains a structural change. In this case the recursive estimates process should have a peak around the change point, whereas the moving estimates (ME) path will have a strong shift.

Similarly fluctuation processes can be computed based on cumulative or moving sums of two types of residuals: the usual OLS residuals or recursive residuals, which are (standardized) one-step ahead prediction errors. The test based on the CUmulative SUM of recursive residuals (the CUSUM test) was first introduced by Brown et al. (1975) and if there is just one structural break in the coefficients the path will start to leave its zero mean around the break point, because the one-step ahead prediction errors will be large. The OLS-based CUSUM and MOSUM (MOving SUM) test have similar properties as the corresponding estimates-based processes and under a single shift alternative the OLS-CUSUM path should have a peak and the OLS-MOSUM path a shift around the change point. **strucchange** offers a unified approach to deal with these processes: given a formula, which specifies a linear regression model, `efp()` computes an empirical fluctuation process of specified type and returns an object of class `"efp"`. The `plot()` method for these objects plots the process path (and preserves the time series properties if the original data was an object of class `"ts"`) by default together with the corresponding boundaries of level $\alpha = 0.05$. The boundaries alone can also be computed by `boundary()`. Finally a significance test, which also returns a p value, can be carried out using the function `sctest()` (structural change test). The proper usage of these functions will be illustrated in the applications section.

F tests

As mentioned in the introduction, F tests are designed to test against a single shift alternative of the

form

$$\beta_i = \begin{cases} \beta_A & (1 \leq i \leq i_0) \\ \beta_B & (i_0 < i \leq n) \end{cases},$$

where i_0 is some change point in the interval $(k, n-k)$. Chow (1960) was the first to suggest a test if the (potential) change point i_0 is known. In his test procedure two OLS models are fitted: one for the observations before and one for those after i_0 and the resulting residuals $\hat{e} = (\hat{u}_A, \hat{u}_B)^\top$ can then be compared with an F test statistic to the residuals $\hat{u}$ from the usual OLS model where the coefficients are just estimated once:

$$F_{i_0} = \frac{(\hat{u}^\top \hat{u} - \hat{e}^\top \hat{e})/k}{\hat{e}^\top \hat{e}/(n-2k)}.$$

For unknown change points (which is the more realistic case) F statistics can be calculated for an interval of potential change points and their supremum can be used as the test statistic. Such a test rejects the null hypothesis if one of the computed F statistics gets larger than a certain critical value or, in other words, if the path of F statistics crosses a constant boundary (defined by the same critical value). The latter shows the possibility to treat sequences of F statistics in a similar way as empirical fluctuation processes: given a formula, which defines a linear regression model, the function `Fstats()` computes a sequence of F statistics for every potential change point in a specified data window and returns an object of class `"Fstats"` (which again preserves the time series properties if the original data had any). Like for `efp` objects there is a `plot()` method available, which plots these F statistics together with their boundary at level $\alpha = 0.05$ or the boundary alone can be extracted by `boundary()`. If applied to `Fstats` objects, `sctest()` computes by default the sup$F$ test statistic and its p value. But there are also two other test statistics available: namely the average of the given F statistics or the exp$F$-functional, which have certain optimality properties (Andrews and Ploberger, 1994).

Application

To demonstrate the functionality of **strucchange** (and to show that there are also applications outside the field of econometrics) we analyze two time series of the number of baptisms (which is almost equivalent to the number of births) and deaths per month in the rural Austrian village Getzersdorf. The data is from the years 1693-1849 (baptisms) and 1710-1841 (deaths) respectively and was collected by the project group "environmental history" from the Institute of Anthropology, Vienna University. The trend of the two time series (extracted by `stl()`) can be seen in Figure 1.

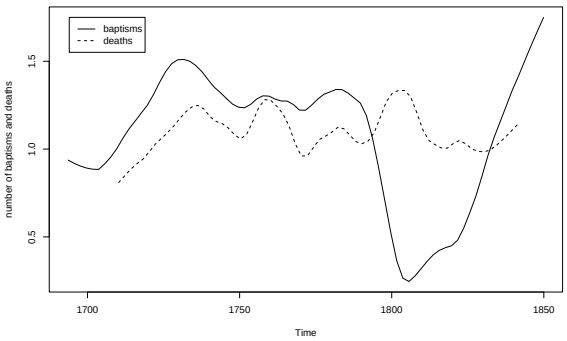

Figure 1: Trend of baptisms and deaths time series from Getzersdorf

We consider the hypothesis that the number of baptisms/deaths remains constant over the sample period. This (almost) implies that the corresponding rate remains constant, because the number of inhabitants remained (almost) constant during the sample period (but is not known explicitely for every month).

The graphs suggest that there was some kind of structural change around 1800 as there is a slight increase in the number of deaths and a dramatic decrease in the number of baptisms. At that time Austria fought against France and Napoleon which explains the decrease of baptisms because the young men were away from home (possibly for several years) and hence couldn't "produce" any offspring.

Analyzing this data with some of the tests from **strucchange** leads to the following results: firstly a Recursive (or Standard) CUSUM model containing just a constant term is fitted to the `ts` objects `baptisms` and `deaths`. The graphical output can be seen in Figure 2.

```
R> baptisms.cus <- efp(baptisms ~ 1,
                       type = "Rec-CUSUM")
R> deaths.cus <- efp(deaths ~ 1,
                     type = "Rec-CUSUM")
R> plot(baptisms.cus); plot(deaths.cus)
```

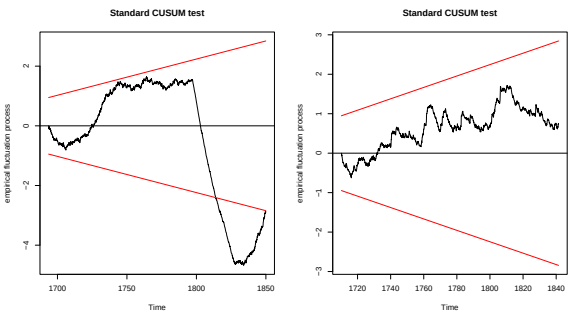

Figure 2: Recursive CUSUM process for baptisms (left) and deaths (right) in Getzersdorf

It can be seen clearly that, whereas the empirical fluctuation process for the death series shows no unusual

behaviour, the CUSUM path for the baptisms starts to deviate from its zero mean around 1800, which indicates a structural change at that time. Furthermore there is some deviation from zero at about 1730 (but which is not significant at the 5% level) which corresponds to the increase in baptisms in the original series. Supplementing this graphical analysis a formal significance test can be carried out and a p value can be computed:

```
R> sctest(baptisms.cus); sctest(deaths.cus)

        Standard CUSUM test

data:  baptisms.cus
S = 1.7084, p-value = 1.657e-05

        Standard CUSUM test

data:  deaths.cus
S = 0.6853, p-value = 0.2697
```

Fitting OLS-MOSUM processes leads to very similar results as Figure 3 shows.

```
R> baptisms.mos <- efp(baptisms ~ 1,
                       type = "OLS-MOSUM")
R> deaths.mos <- efp(deaths ~ 1,
                     type = "OLS-MOSUM")
R> plot(baptisms.mos); plot(deaths.mos)
```

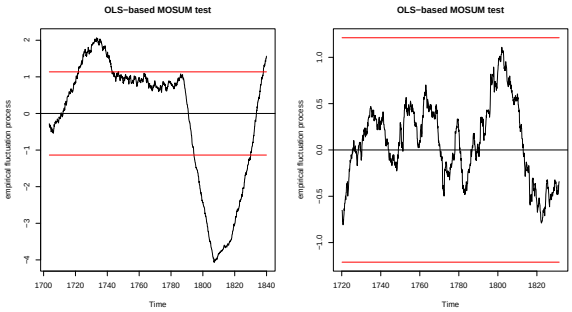

Figure 3: OLS-based MOSUM process for baptisms (left) and deaths (right) in Getzersdorf

The fluctuation of the deaths process remains within its boundaries, although there is a non-significant shift at about 1800. The MOSUM path for the baptisms on the other hand has two shifts: a smaller one around 1730 and a stronger one at 1800, which emphasizes the Recursive CUSUM results.

Finally F statistics are computed for the given times series and the results can be seen in Figure 4.

```
R> baptisms.Fstats <- Fstats(baptisms ~ 1)
R> deaths.Fstats <- Fstats(deaths ~ 1)
R> plot(baptisms.Fstats); plot(deaths.Fstats)
```

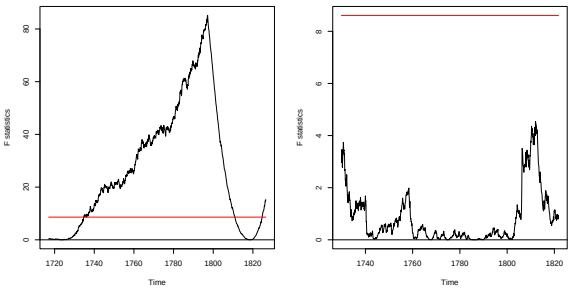

Figure 4: F statistics for baptisms (left) and deaths (right) in Getzersdorf

As in the generalized fluctuation tests no significant result can be achieved for the deaths series, although there is a small peak around 1810, whereas there is an overwhelmingly significant peak at around 1800 for the baptisms. Note that the F statistics just detect the stronger shift at 1800, because they were designed for single shift alternatives.

Summary

strucchange offers a unified framework for generalized fluctuation and F tests for structural change and it extends common significance tests by means to visualize the data and to identify structural changes. More detailed information about the features of **strucchange** can be found in Zeileis et al. (2001).

Bibliography

D. W. K. Andrews. Tests for parameter instability and structural change with unknown change point. *Econometrica*, 61:821–856, 1993. 8

D. W. K. Andrews and W. Ploberger. Optimal tests when a nuisance parameter is present only under the alternative. *Econometrica*, 62:1383–1414, 1994. 9

R. L. Brown, J. Durbin, and J. M. Evans. Techniques for testing the constancy of regression relationships over time. *Journal of the Royal Statistical Society*, B 37:149–163, 1975. 8

G. C. Chow. Tests of equality between sets of coefficients in two linear regressions. *Econometrica*, 28: 591–605, 1960. 9

B. E. Hansen. Tests for parameter instability in regressions with I(1) processes. *Journal of Business & Economic Statistics*, 10:321–335, 1992. 8

C.-M. Kuan and K. Hornik. The generalized fluctuation test: A unifying view. *Econometric Reviews*, 14: 135–161, 1995. 8

A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. strucchange: An R package for testing for structural change in linear regression models. Report 55, SFB "Adaptive Information Systems and Modelling in Economics and Management Science", May 2001. URL http://www.wu-wien.ac.at/am/reports.htm#55. 10

Achim Zeileis
Technische Universität Wien, Austria
zeileis@ci.tuwien.ac.at

Programmer's Niche: Macros in R

Overcoming R's virtues

by Thomas Lumley

A familiar source of questions on the R mailing lists is the newly converted R user who is trying to write SAS or Stata code in R. Bill Venables then points out to them that R is not a macro language, and gently explains that there is a much easier solution to their problems. In this article I will explain what a macro is, why it's good that R isn't a macro language, and how to make it into one.

There are two reasons for this. It has been famously observed[1] that a Real Programmer can write Fortran code in any language, and it is similarly an interesting exercise to see how R can implement macros. Secondly, there are a few tasks for which macros are genuinely useful, which is why languages like LISP, for example, provide them.

What is a macro language?

Suppose you have a series of commands

```
table(treatment, gender)
table(treatment, race)
table(treatment, age.group)
table(treatment, hospital)
table(treatment, diabetic)
```

These commands can be created by taking the skeleton

```
table(treatment, variable)
```

substituting different pieces of text for `variable`, and evaluating the result. We could also repeatedly call the `table()` function with two arguments, the first being the values of `treatment` and the second being the values of the other variable.

R takes the latter approach: evaluate the arguments then use the values. We might define

```
rxtable <- function(var){
  table(treatment, var)
}
```

Stata typically takes the former approach, substituting the arguments then evaluating. The 'substitute then evaluate' approach is called a *macro expansion*, as opposed to a *function call*. I will write this in pseudo-R as

```
rxtable <- macro(var){
  table(treatment, var)
}
```

Why not macros?

In this simple example it doesn't make much difference which one you use. In more complicated examples macro expansion tends to be clumsier. One of its advantages is that you get the actual argument names rather than just their values, which is useful for producing attractive labels, but R's lazy evaluation mechanism lets you do this with functions.

One problem with macros is that they don't have their own environments. Consider the macro

```
mulplus <- macro(a, b){
  a <- a+b
  a * b
}
```

to compute $(a+b)(b)$. This would work as a function, but as a macro would have undesirable side-effects: the assignment is not to a local copy of `a` but to the original variable. A call like `y <- mulplus(x, 2)` expands to `y <- {x<-x+2; x*2}`. This sets `y` to the correct value, $2x+4$, but also increments `x` by 2. Even worse is `mulplus(2, x)`, which tries to change the value of 2, giving an error.

We could also try

```
mulplus <- macro(a, b){
 temp <- a+b
 temp * b
}
```

This appears to work, until it is used when we already have a variable called `temp`. Good macro languages need some way to provide variables like `temp` that are guaranteed not to already exist, but even this requires the programmer to declare explicitly which variables are local and which are global.

The fact that a macro naturally tends to modify its arguments leads to one of the potential uses of macro expansion in R. Suppose we have a data frame

[1] "Real Programmers don't use Pascal" by Ed Post — try any web search engine