# The “StrucBreak” package for gretl*

**Riccardo Lucchetti and Sven Schreiber**

version 0.9, October 2020

## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General comments</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Usage</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Installation</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Script use</td>
<td>3</td>
</tr>
<tr>
<td>2.3</td>
<td>GUI use</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Alphabetical list of public functions</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Members of the central bundle</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>Scalars</td>
<td>9</td>
</tr>
<tr>
<td>4.2</td>
<td>Matrices</td>
<td>10</td>
</tr>
<tr>
<td>4.3</td>
<td>Series</td>
<td>10</td>
</tr>
<tr>
<td>4.4</td>
<td>Strings</td>
<td>11</td>
</tr>
<tr>
<td>4.5</td>
<td>String arrays</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>To do</td>
<td>11</td>
</tr>
<tr>
<td>A</td>
<td>Example</td>
<td>12</td>
</tr>
<tr>
<td>A.1</td>
<td>Script</td>
<td>12</td>
</tr>
<tr>
<td>A.2</td>
<td>Output of the example script</td>
<td>13</td>
</tr>
<tr>
<td>B</td>
<td>Changelog</td>
<td>16</td>
</tr>
</tbody>
</table>

## 1 General comments

This package serves to (a) test for and (b) estimate dates of structural breaks, and (c) estimate other parameters in the presence of such breaks in a single time-series regression equation. The main practical reference is: Bai J. & Perron P. (2003), Computation and analysis of multiple structural change models, Journal of Applied Econometrics 18: 1-22. See also references therein.

*The best way to give feedback or ask questions is to use the gretl users mailing list, see https://gretlml.univpm.it/postorius/lists/gretl-users.gretlml.univpm.it/. Authorship for the package itself (as opposed to this help document) is shared between Riccardo Lucchetti, Sven Schreiber, and Giulio Palomba."
Based on unpublished Ox code by Riccardo Jack Lucchetti and Giulio Palomba, the initial port to hansl was done by Sven Schreiber. The new name is “StrucBreak” or in short form “SB”, following the example of the DPB or gig packages in gretl which also get their names from the subject matter, not from certain persons.

As long as the versions numbers are below 1.0 in principle the interface might change in future releases, although we will try to avoid this as much as possible.

## 2 Usage

The StrucBreak package can be used by writing hansl scripts as well as via the dialog window that gretl provides for contributed packages. As of version 0.9 there is now also a GUI wrapper function that is hopefully mostly self-explanatory; see Figure 1. An important point about notation which can also be seen in that Figure: The regressors with changing coefficients are denoted with $Z(t)$ and those where the coefficients are forced to be the same with $X(t)$. The non-breaking $X$ coefficients are called $\beta$.

There are several example scripts that come bundled with the package: First there’s
the mandatory and official sample script. This works with artificial data. Then there are the scripts in the example sub-directory which are named BP2003_tableII.inp and BP2003_tableIII.inp. Here we focus on the steps of the first of these example scripts based on Bai & Perron (2003).

2.1 Installation

Get it from the gretl package server in the usual way: Either via menus Tools / Function packages / On Server, select StrucBreak and click install, or on the console or command line by doing:

pkg install StrucBreak

2.2 Script use

Let us look at each line in the first example script. The script mimics the calculations related to Table II in the Bai & Perron (2003, J of Appl. Econometrics, henceforth BP) paper. However, notice that the published results are not the current state of the art, later versions of Pierre Perron’s own Gauss code produce slightly different results. Thus the aim here is not to replicate the published table 1-to-1.

First of all, we have the include statement that loads the package:

include StrucBreak.gfn

which always has to come first. Next, we load the relevant dataset that BP used (which is distributed with the package, a fact that we have to communicate to gretl):

open uk.gdt --frompkg=StrucBreak

Now we have to prepare the regressors that are considered to be subject to breaks. In the present example this includes the constant term (which has to be added explicitly, for example because it could also be non-breaking instead) and the lagged inflation rate. The needed argument type in the package is a gretl list of series.

list Z = const infl(-1)

Notice that the name of the list is arbitrary, but here we aim at being as close as possible to BP’s original notation. Another list to be specified concerns the regressors with constant, non-breaking coefficients, which however in this case is empty:

list X = null

Of course, the explicit creation of an empty list is not strictly necessary, you could also type the null keyword directly as the corresponding argument in the call to SB_Setup() below, in place of “X”.

Now we want to specify the same sample that BP used:

smpl 1948 1987

Also, we want to define a particular amount of trimming, namely a fraction which corresponds to 8 observations in this sample size. Hence we perform this auxiliary calculation to be used in the next step:

trim8 = 8 / $nobs
This concludes the first round of preparations, and we’re ready to really dive into the StrucBreak package. We could start by telling the package our basic specification, using the `SB_Setup()` function as follows:

```gretl
bundle bII = SB_Setup(infl, Z, X)
```

The first argument must be a gretl series and defines the dependent (left-hand side) variable. Then we have the lists of breaking and non-breaking terms, respectively.

However, here we will at the same time pass a number of additional options as part of the setup call. These option value assignments have to be wrapped in a gretl bundle, and so the full setup call in this example script is given like this:

```gretl
bundle bII = SB_Setup(infl, Z, X, defbundle("MaxBreaks",3, "trimming",trim8, "Hetdat",1, "Hetomega",0, "Hetq",0, "Hetvar",1)))
```

The `MaxBreaks` key gives a positive integer of the maximum number of breaks considered in the sample, the default value being data-dependent (since v0.9) by leaving an average of at about 50 observations between each break.

The key “trimming” must be between 0 and 1; it indicates the minimum fraction of the total sample size separating two consecutive breakpoints. For example, suppose you have a quarterly dataset spanning 20 years, which amounts to 80 observations. A trimming parameter set to 0.075 would mean that consecutive breaks cannot occur closer than 6 quarters from one another. This argument is optional, too, with a default value of 0.1. The other parameters will be explained in a minute.

The distance between breaks $h$ is internally calculated as the sample size times the trimming parameter, and you get a warning if the sample size, the maximum number of breaks, and the trimming are not compatible. To avoid breakage you should not manually adjust the distance between breaks, instead change the trimming if you must.

The `SB_Setup()` call returns a gretl bundle as the central container of all the needed bits and pieces for a concrete StrucBreak application. All further testing and estimation commands operate on this specific bundle and thus you need to pass this bundle to all the subsequent functions. For performance reasons this passing is done without copying, and therefore it is done in pointer form. For the user this just means to type the ampersand sign “&” before the bundle name, as is shown below.

```gretl
There are a few other settings that are also given their default values through the `SB_Setup()` call and which can be manipulated as in the example script; see Table 1. Note that setting `Prewhit` to 1 also forces `Robust` to 1.

There are actually various ways how you can change these settings apart from including them already in the setup call. First of all you can simply manipulate such a bundle element after the setup directly, for example:

```gretl
b.Hetq = 0
```

Secondly you can collect various options in a new temporary bundle and thereby override the previous values as follows:

```gretl
bundle myoptions = defbundle("Robust",1, "PrintIter",1)
b = myoptions + b # ordering is important!
```

1In very recent gretl versions (as of late 2020) you can use the convenience function `()` and write the following first line in a different way: `bundle myoptions = _{Robust=1, PrintIter=1}`
<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robust</td>
<td>0 (FALSE)</td>
<td>HAC-robust variance for the parameters</td>
</tr>
<tr>
<td>Prewhit</td>
<td>0 (FALSE)</td>
<td>Use VAR(1) prewhitening for the HAC covariance matrix?</td>
</tr>
<tr>
<td>Hetdat</td>
<td>1 (TRUE)</td>
<td>Is the DGP for all the regressors considered possibly nonconstant across regimes?</td>
</tr>
<tr>
<td>Hetvar</td>
<td>1 (TRUE)</td>
<td>Is $\sigma^2$ considered possibly nonconstant across regimes?</td>
</tr>
<tr>
<td>Hetomega</td>
<td>1 (TRUE)</td>
<td>Is $\Omega$ considered possibly nonconstant across regimes?</td>
</tr>
<tr>
<td>Hetq</td>
<td>1 (TRUE)</td>
<td>Is $E(z_t z_t')$ considered possibly nonconstant across regimes?</td>
</tr>
<tr>
<td>PrintIter</td>
<td>0 (FALSE)</td>
<td>print iteration details (quite verbose!)</td>
</tr>
</tbody>
</table>

Note: The literal aliases TRUE and FALSE can be used from gretl 2020b onwards.

Table 1: Boolean options to a SB bundle

Or there is the old variant from previous (< 0.9) package versions where you can change these on/off settings in a slightly more verbose way by defining a gretl array of strings² where each of the included strings contains the name of the option first and then immediately “Yes” or “No” (case sensitive). For example:

```
strings opts = defarray("HetdatYes", "HetvarYes", "HetomegaNo", "HetqNo")
```

In this latter case, to make your desired options known you have to call the special `SB_SetOptions()` function with the pre-existing model bundle (in pointer form as mentioned before):

```
SB_SetOptions(&bII, opts)
```

This method is basically kept for now to remain backwards-compatible, but is no longer recommended and may be dropped in the future.

Note that, at any stage of your script, you can query the current values of the options by calling the `SB_PrintOptions(bII)` function.

Finally, we are in the position to perform the actual tests and estimates. This is done by calling one or more of the following functions, always with the constructed model bundle as the first and often only argument. These functions do not directly return any objects, but instead they provide all their results in the form of additional members of the model bundle, see section 4 for a listing.

The first possibility is to invoke a global optimization procedure that finds the location of breaks for the different possible numbers of breaks (up to the max. specified number):³

```
SB_Global(&bII)
```

Based on this, some tests could be run with the following function, including the $supF(l|0)$ and the WDmax and UDmax tests that take a break-free model as the baseline:

```
SB_Tests(&bII)
```

²Even if it just holds a single string, it then must be defined as a 1-element array of strings, not as a gretl string type.
³If for some reason you want to use specific initial values for the estimation algorithm of the $\beta$ coefficients, you can inject a suitable column vector into the model bundle under the label “Betaini” before proceeding.
For example, at this point you could inspect the estimated break dates manually by looking at the `BreakDates` bundle member (which in the example script is skipped, however, and thus commented out):

```c
# eval bII.BreakDates
```

One could also use the following sequential procedure which uses \( supF(l + 1|l) \) tests, but which does not require to run the global minimizing procedure before.

```c
SB_Sequ(&bII)
```

Once the preferred number of breaks is determined, you can estimate the corresponding model with breaks like this, where the third argument is the wanted number of breaks:

```c
SB_Estimate(&bII, ''FIX'', 2)
```

See the documentation of the `SB_Estimate` function for other estimation options. The estimation call also prints out the estimation results, so the special function is redundant here and is commented out:

```c
# SB_PrintEstimates(bII)
```

But sometimes it may be convenient to pick a model bundle from a StrucBreak application on which estimation had already be performed, and then this function can be used to see the earlier results. Apart from printing them out, in more complex application scenarios you can extract from the model bundle (here `&bII`) the results of the tests and estimates as gretl objects (mainly matrices), see section 4.

### 2.3 GUI use

The function attaches itself to the Univariate Time Series submenu in the Model menu (in the gretl main window). Or as usual you can execute it from the window that lists all installed function packages. For most of the entry fields and options it should be clear from the labels and the listing in this help document to which parameters and settings they correspond.

The most complex user setting concerns the two drop-down list items “action” and “breaks in estimation based on...”. Not all combinations there are actually mutually compatible, but the package should catch those cases.

FIXME before v1.0: Expand explanation.

### 3 Alphabetical list of public functions

All these functions carry the prefix “SB_”, so that in your code applications it’s easy to see where they’re coming from.

```c
SB_Estimate(bundle *b, string method[null], int n[0::0], bool verbosity[1])
```

Estimates the model. Specify which method should determine the underlying number of breaks and their location (dating) in the second argument as a string – alternatively this string can be provided as the bundle member `b.method`, manually added to
the bundle before this function is called. The corresponding procedure must have been run before (mainly \texttt{SB\_Global} or \texttt{SB\_Sequ}). The results are printed out unless verbosity is set to 0.

The number of breaks to be imposed needs to be passed as \texttt{n} only if the method \texttt{“FIX”} is chosen.

Returns scalar (just indicates success if 0). Its arguments are:

1. \texttt{b}: a model bundle (in pointer form) \textbf{(required)}
2. \texttt{method}: string to specify break determination method (one of \texttt{“BIC”}, \texttt{“LWZ”}, \texttt{“SEQ”}, \texttt{“REP”}, \texttt{“FIX”}; optional, if instead \texttt{b.method} exists)
3. \texttt{n}: integer to specify number of breaks for the \texttt{“FIX”} method (optional, except if \texttt{method==’FIX’})

\begin{verbatim}
SB\_Global(bundle \*b)
\end{verbatim}

Calls the procedure to obtain the break dates and the associated sum of squared residuals for all numbers of breaks between 1 and \texttt{m}. (Break dates in \texttt{b.BreakDates}, the other results internal.) Also invokes the calculation of the information criteria BIC and LWZ (Liu, Wu and Zidek) and the corresponding suggested number of breaks. It returns (=adds them to the bundle) the two selected break numbers, as \texttt{b.BICBreaks} and \texttt{b.LWZBreaks}.

Returns a scalar (just indicates success if 0). Argument:

1. \texttt{b}: a model bundle (in pointer form) \textbf{(required)}

\begin{verbatim}
SB\_PrintEstimates(bundle b, string meth[null])
\end{verbatim}

Prints the estimates based on the choice method, which must be one of \texttt{“BIC”} (global procedure based on BIC criterion), \texttt{“LWZ”} (global based on LWZ), \texttt{“SEQ”} (sequential procedure), \texttt{“REP”} (sequential procedure followed by repartition), or \texttt{“FIX”} (use a given number of breaks). The corresponding estimates must have been produced before and thereby stored in the bundle. The default is to print everything there is.

Returns scalar (just indicates success if 0). Argument:

1. \texttt{b}: a model bundle (not in pointer form) \textbf{(required)}
2. \texttt{meth}: results from which estimation method (optional)

\begin{verbatim}
SB\_PrintOptions(bundle b)
\end{verbatim}

Prints out the currently set options for the model contained in bundle \texttt{b}. Returns nothing. Arguments:

1. \texttt{b}: a model bundle \textbf{(required)}
SB_Sequ(bundle *b, bool doREP[1])

Calls the sequential procedure that estimates each break one at a time. It stops when the supF(l + 1/l) test is not significant. It returns (= adds them to the bundle) the number of breaks found (b.SEQBreaks) and the break dates (b.SEQDates). Note that it can be used independently of the other procedures, i.e. global minimizers need not be obtained since it used a method to compute the breaks in O(T) operations.

By default (doREP==1) also constructs the so-called repartition updated estimates of the break dates obtained by the sequential method; see Bai (1995), Estimating breaks one at a time, Econometric Theory, 13, 315-352.

It allows estimates that have the same asymptotic distribution as those obtained by global minimization. Otherwise, the output from the procedure “c_estim” (private) does not deliver asymptotically correct confidence intervals for the break dates. The result is provided in b.REPDates (the first-step results of the sequential procedure in b.SEQDates remain untouched and available).

Returns scalar (just indicates success if 0). Argument:

1. b: a model bundle (in pointer form) (**required**)
2. doREP: boolean switch whether to invoke the repartition procedure (optional)

SB_SetOptions(bundle *b, strings opts)

Sets or overrides settings, no returns. Its arguments are:

1. b: a model bundle (in pointer form) (**required**)
2. opts: an array of strings (usage of the defarray() function recommended); recognized options are: “Robust”, “Prewhit”, “Hetdat”, “Hetvar”, “Hetomega”, “Hetq”, “PrintIter”. (**required**)

Usage example:

```
SB_SetOptions(&MyModel, defarray("RobustYes", "PrewhitNo"))
```

SB_Setup(series y, list Z, list X, bundle bopts[null])

Returns a bundle to be passed around subsequently. For the allowed elements of the option bundle see elsewhere in this help document. Its arguments are:

1. y: a series containing $y_i$, the dependent variable of the model (**required**)
2. Z: a list containing $p$ regressors whose coefficients are allowed to change across regimes (includes deterministics such as the constant if necessary) (**required**)
3. X: a list containing the variables whose coefficients are assumd to be constant across regimes (default: empty)
4. bopts: bundle with option values (default: empty, the parameter defaults are not overridden)

SB_Tests(bundle *b, bool doSupFseq[1])

Calls the procedure to perform and print various tests for the number of breaks, including the UDmax and WDmax tests, and the SupF tests against fixed break numbers. (Adds results to the bundle.)

By default (doSupFseq==1) also calls the procedure that performs the \( supF(l + 1|l) \) tests where the first \( l \) breaks are taken from the global minimization.

Returns scalar (just indicates success if 0). Its argument is:

1. b: a model bundle (in pointer form) (required)
2. doSupFseq: boolean switch whether to run the sequential SupF tests as well (optional)

4 Members of the central bundle

The central bundle is created by calling the SB_Setup() function. Some bundle members are added only later. The boolean settings were already enumerated in Table 1.

4.1 Scalars

With defaults if applicable.

- b.trimming (0.1) – trimming parameter\(^4\)
- b.MaxBreaks (data-dependent default: one break after around 50 obs on average\(^5\))
  - maximum number of breaks considered
- b.T – number of obs in sample
- b.t1 – index of first obs
- b.t2 – index of final obs
- b.nh – minimum number of obs between consecutive breaks (directly implied by the amount of trimming; do not try to alter this yourself)
- b.CurBreaks – number of breaks currently used for estimation
- b.BICBreaks – number of breaks chosen by the BIC criterion
- b.LWZBreaks – number of breaks chosen by the LWZ criterion
- b.q – number of Z elements

\(^4\)Prior to v0.9 this was called Eps1.
\(^5\)Prior to v0.9 the default was 3.
• \( b.p \) – number of X elements
• \( b.\text{MaxIter} = 20 \) – internal iteration limit
• \( b.Eps = 0.00001 \) – <internal convergence criterion>

4.2 Matrices
• \( b.\text{SEQBreaks} \) (col vector with length 4 [number of hardwired significance levels]) – suggested numbers of breaks of the sequential procedure for each significance level
• \( b.\text{my} \) – dependent variable
• \( b.nX \) – regressors with non-breaking coefficients (may be empty)
• \( b.nZ \) – regressors with changing coefficients
• \( b.\text{BreakDates} \) – quasi-triangular matrix collecting the break dates for each of the possible break numbers (one date for one break in 1st col, two dates for two breaks in 2nd col, etc.) from the global procedure.
• \( b.\text{SEQDates} \) – matrix for the break dates determined by the sequential procedure at various significance levels (signif. levels in rows)
• \( b.\text{REPDates} \) – similar matrix for the break dates determined by the repartition procedure
• \( b.\text{SupFDates} \) – vector for the break dates corresponding to the SupF tests
• \( b.\text{SupFTest} \) – vector of tests statistics for \( l + 1 \) against \( l \) breaks, for all possible \( l \)
• \( b.\text{Siglev} \) – hardcoded significance levels, corresponding to the CV functions that only contain the associated values
• \( b.\text{Coef} \) – coefficients of the estimated model
• \( b.\text{VCV} \) – covariance matrix of the estimated coefficients
• \( b.\text{BreaksCI} \) – matrix of the confidence intervals of the break dates (two significance levels: 95% lower bound in col 1 and upper in col 2, and 90% lower bound in col 3 and upper in col 4)
• \( b.\text{BigMat} \) – <internal>
• \( b.\text{Betaini} \) – optional user-supplied initial values for the X coefficients
• \( b.\text{global} \) – <internal>

4.3 Series
• \( b.y \) (Not needed internally because work is done on \( b.\text{my} \). But the user could grab it from the bundle later.)
4.4 Strings

- \texttt{b.yname} – name of the dependent variable
- \texttt{b.method} – Can be set manually to influence the subsequent estimation. Must be one of “BIC”, “LWZ”, “SEQ”, “REP” (case sensitive).

4.5 String arrays

- \texttt{b.Xnames} – names (one string per name) of the non-breaking variables
- \texttt{b.Znames} – names of the variables with changing coefficients

5 To do

Don’t hold your breath for these features, some may never be implemented. The more demand there is, the more probable that somebody might work on it.

- LR-test based confidence sets (Eo, Y., Morley, J. (2015). Likelihood-ratio-based confidence sets for the timing of structural breaks. Quantitative Economics 6:463–497); see also Chang & Perron (2015, A comparison of alternative methods to construct confidence intervals for the estimate of a break date in linear regression models, Econometric Reviews, http://dx.doi.org/10.1080/07474938.2015.1122142): “On the basis of achieving an exact coverage rate that is closest to the nominal level, Elliott and Mueller’s (2007) approach is by far the best one. However, this comes with a very high cost in terms of the length of the confidence intervals.”

- Multi-equation setup (Qu & Perron, 2007)
A  Example

This example script generates an artificial data set with 200 observations according the following specification:

\[ y_t = \beta_0 + \beta_1 z_t + \beta_2 x_t + \epsilon_t \]

where

\[ \beta_0 = \begin{cases} 1 & \text{for } t \leq 150, \\ -1 & \text{otherwise} \end{cases} \quad (1) \]
\[ \beta_1 = \begin{cases} 1 & \text{for } t \leq 100, \\ -1 & \text{otherwise} \end{cases} \quad (2) \]
\[ \beta_2 = 1 \quad (3) \]
\[ V(\epsilon_t) = 1 \quad (4) \]

and runs the corresponding model.

A.1  Script

# StrucBreak example on artificial data
set verbose off
include StrucBreak.gfn
set seed 732237
nulldata 240

# declare as monthly dataset
setobs 12 1995:1

series e = normal()
series x = normal()
series z = normal()

# subsample 1
smpl ; 1999:12
series y = 1 + x + z + e

# subsample 2
smpl 2000:1 2009:12
series y = 1 + x - z + e

# subsample 3
smpl 2010:1 2014:12
series y = -1 + x - z + e

# Now turn to specify model to estimate on this data
list Z = const z
list X = x
smpl full

bundle Model = SB_Setup(y, Z, X)
# modify some settings in this Model bundle
bundle bopts = defbundle("MaxBreaks",3, "trimming",$nobs*0.2), "Hetvar",1)
Model = bopts + Model

SB_Global(&Model)
SB_Tests(&Model)

# SB_SupFTest(&Model) # (run by default from SB_Tests)
SB_Sequ(&Model)

SB_Estimate(&Model, "FIX", 2)       # with fixed number of breaks
# SB_PrintEstimates(Model) # (run by default from SB_Estimate)

A.2 Output of the example script

StrucBreak 0.9, 2020-09-30 (Riccardo "Jack" Lucchetti and Sven Schreiber)
SB_Setup: OK!
Warning: forcing option HETDAT as p>0

===================================================================
Options are now:
===================================================================
Robust S.E.: OFF
Warning: inappropriate if lagged dependent variables
are included as regressors
Hetdat: ON
Hetvar: ON, Hetomega: ON, Hetq: ON
beta: Nonfixed
Eps = 0.000 trimming = 0.200 Max. breaks = 3
Min. distance between breaks = 48

===================================================================
OUTPUT FROM THE GLOBAL OPTIMIZATION STAGE
===================================================================
This is a partial structural change model and the following
specifications were used:
Number of regressors with fixed coefficients: 1
The convergence criterion is: 0.000010
The iterations will not be printed

-----------------------------------------------
Breaks    SSR     Dates
    1   370.18422  181 (2010:01)
    2  223.50144   60 (1999:12), 181 (2010:01)

===================================================================
OUTPUT FROM THE APPLICATION OF INFORMATION CRITERIA
===================================================================

Breaks   BIC    lwz
     0 0.8414  0.8498
The number of breaks chosen by BIC is: 2
The number of breaks chosen by LWZ is: 2

OUTPUT FROM THE TESTING PROCEDURES

a) supF tests against a fixed number of breaks

\[
\begin{align*}
\text{supF}(1|0) & \quad \text{supF}(2|0) & \quad \text{supF}(3|0) \\
156.872 & \quad 166.195 & \quad 110.696
\end{align*}
\]

Critical values:

\[
\begin{align*}
\text{supF}(1|0) & \quad \text{supF}(2|0) & \quad \text{supF}(3|0) \\
10\% & \quad 9.37 & \quad 7.91 & \quad 6.43 \\
5\% & \quad 10.98 & \quad 8.98 & \quad 7.13 \\
2.5\% & \quad 12.59 & \quad 10.00 & \quad 7.92 \\
1\% & \quad 14.92 & \quad 11.30 & \quad 8.95
\end{align*}
\]

b) Dmax tests against an unknown number of breaks

UDmax test: 166.195451
Crit. values: 10%: 9.66 | 5%: 11.16 | 2.5%: 12.68 | 1%: 14.92 |

WDmax test (crit. val.)

\[
\begin{align*}
\text{supF}(2|1) & \quad \text{supF}(3|2) \\
196.87 & \quad 203.21 & \quad 209.24 & \quad 219.44
\end{align*}
\]

Critical values: 10% 5% 2.5% 1%

\[
\begin{align*}
\text{supF}(2|1) & \quad \text{supF}(3|2) \\
10.92 & \quad 12.55 & \quad 14.22 & \quad 16.69 \\
11.90 & \quad 13.46 & \quad 15.39 & \quad 17.41
\end{align*}
\]

OUTPUT FROM THE SEQUENTIAL PROCEDURE

At 10.0% sig. level...
the first break found is at: 181 (2010:01)
the next break found is at: 60 (1999:12)
... a total of 2 breaks
At 5.0% sig. level...
the first break found is at: 181 (2010:01)
the next break found is at: 60 (1999:12)
... a total of 2 breaks
At 2.5% sig. level...
the first break found is at: 181 (2010:01)
the next break found is at: 60 (1999:12)
... a total of 2 breaks

At 1.0% sig. level...
the first break found is at: 181 (2010:01)
the next break found is at: 60 (1999:12)
... a total of 2 breaks

Repartition procedure for the 10.0% significance level:
-------------------------------------------------------------
The updated break dates are:
60 (1999:12) 181 (2010:01)

Repartition procedure for the 5.0% significance level:
-------------------------------------------------------------
...dates are the same as for the 10.0% level

Repartition procedure for the 2.5% significance level:
-------------------------------------------------------------
...dates are the same as for the 5.0% level

Repartition procedure for the 1.0% significance level:
-------------------------------------------------------------
...dates are the same as for the 2.5% level

OUTPUT FROM ESTIMATION OF THE MODEL WITH 2 BREAKS (fixed)
===================================================================
Dependent variable: y
-------------------------------------------------------------------
Unbreakables

<table>
<thead>
<tr>
<th>Koeffizient</th>
<th>Std.-fehler</th>
<th>z</th>
<th>p-Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1.01879</td>
<td>0.0643043</td>
<td>15.84</td>
</tr>
</tbody>
</table>

Subsample 1: 1995:01 - 1999:12

<table>
<thead>
<tr>
<th>Koeffizient</th>
<th>Std.-fehler</th>
<th>z</th>
<th>p-Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>const</td>
<td>1.09134</td>
<td>0.130655</td>
<td>8.353</td>
</tr>
<tr>
<td>z</td>
<td>0.927268</td>
<td>0.129479</td>
<td>7.162</td>
</tr>
</tbody>
</table>

Subsample 2: 2000:01 - 2010:01

<table>
<thead>
<tr>
<th>Koeffizient</th>
<th>Std.-fehler</th>
<th>z</th>
<th>p-Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>const</td>
<td>0.993910</td>
<td>0.0846317</td>
<td>11.74</td>
</tr>
<tr>
<td>z</td>
<td>-1.01908</td>
<td>0.0899632</td>
<td>-11.33</td>
</tr>
</tbody>
</table>

Subsample 3: 2010:02 - 2014:12

<table>
<thead>
<tr>
<th>Koeffizient</th>
<th>Std.-fehler</th>
<th>z</th>
<th>p-Wert</th>
</tr>
</thead>
</table>
Confidence intervals for the break dates

The 95% C.I. for the 1th break is: 1999:09 - 2000:03
The 90% C.I. for the 1th break is: 1999:10 - 2000:03
The 95% C.I. for the 2th break is: 2009:10 - 2010:04
The 90% C.I. for the 2th break is: 2009:11 - 2010:03

B Changelog

• 0.9, October 2020
  Attention, the usage of the SB_Setup function has changed along with the way how to specify options! You need to adjust your scripts.
  Add GUI interface function SB_GUI; fix stupid leftover bugs in SB_EstimCrit and also quite a few more input error checks; plus some more off-by-one bugs still from the port of 0-based Ox; another but small incompatible change: SB_PrintEstimates and SB_PrintOptions now take a plain bundle argument, not in pointerized form; try to avoid redundant calculations and output for the sequential and repartition routines; drop the misleading Fixb option (which only meant user-provided initialization); marginal efficiency improvement in the NLDat internal function; actually enable the PrintIter choice independently of an internal setting

• 0.5, September 2020 (no public release): internally use the native lrcovar() function, require gretl 2018b because of that

• 0.21, January 2018: fix off-by-one bug leading to wrong critical values (printout only) for the SupF sequential tests; also, a few cosmetic updates

• 0.2, February 2017: fix gretl syntax warning, minor help doc update

• 0.1, November 2016: initial public release