The "StrucBreak" package for gretl*

Riccardo Lucchetti and Sven Schreiber

somewhat preliminary version 0.21, January 2018

Contents

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.

Based on unpublished Ox code by Riccardo Jack Lucchetti and Giulio Palomba, the initial port to hansl was done by Sven Schreiber.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.

*The best way to give feedback or ask questions is to use the gretl users mailing list, see http://lists.wfu.edu/mailman/listinfo/gretl-users. Authorship for the package itself (as opposed to this help document) is shared between Riccardo Lucchetti, Sven Schreiber, and Giulio Palomba.

The first public release 0.1 version was in November 2016, and as long as the versions are called preliminary the interface might change in future releases.

2 Usage

It is planned that the StrucBreak package could be used by writing hansl scripts as well as via the dialog windows that gretl provides for contributed packages. However, so far only the hansl scripting interface is ready, the GUI wrapper functions still need to be added. Therefore we describe only the scripting usage in this version of the documentation.

We also focus on the steps of the bundled example script, the documentation is not complete yet.

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:

```
install StrucBreak
```

2.2 Script use

Let us look at each line in the included 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):

```
open uk.gdt
```

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
```

This concludes the first round of preparations, and we're ready to really dive into the StrucBreak package. We start by telling the package our basic specification, using the `SB_Setup()` function:

```
bundle bII = SB_Setup(infl, Z, X, 3, 0.2)
```

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. The fourth argument is optional (at least if the final fifth argument were omitted) and gives a non-negative integer of the maximum number of breaks considered in the sample, the default value being 3.[1]

The final argument concerns the amount of trimming (`Eps1` internally), which varies between 0 and 1; it indicates the minimum fraction of the total sample size separating two consecutive brakpoints. 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 distance between breaks `nh` 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. But you can in principle manually adjust the distance between breaks after the `SB_Setup()` call, as we do here:

```
bII.nh = 8 # distance between breaks
```

However, by doing this it is the user's responsibility to specify a feasible value, otherwise strange things might happen.

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.

There is a bunch of other settings that are given their default values through the `SB_Setup()` call; see Table 1. Note that setting `Prewhit` to 1 also forces `Robust` to 1.

You can change these on/off settings by defining a gretl array of strings,[2] where each of the included strings contains the name of the option first and then immediately "Yes" or "No" (case sensitive). The example script contains an –guess what– example of this:

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

To make your desired options known to the current application of the StrucBreak package, 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)
```

[1] The value 0 (no breaks) is allowed, but it'd be rather pointless if you're using the StrucBreak package. If for some reason you want to change this value after you have already called `SB_Setup()`, you would have to change the internal variable `bII.MaxBreaks`, where `bII` is the chosen name of the model bundle.

[2] Even 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.

Name	Default Value	Description
Robust	0	HAC-robust variance for the parameters
Prewhit	0	Use VAR(1) prewhitening for the HAC covariance matrix?
Hetdat	1	Is the DGP for all the regressors considered possibly nonconstant across regimes?
Hetvar	1	Is σ^2 considered possibly nonconstant across regimes?
Hetomega	1	Is Ω considered possibly nonconstant across regimes?
Hetq	1	Is $E(z_t z_t')$ considered possibly nonconstant across regimes?
Fixb	0	
PrintIter	0	

Table 1: Boolean options to a SB bundle

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):

```
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)
```

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):

```
# 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.

```
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:

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

See the documentation of the `SB_Estimate` function for other estimation options.

Finally, presumably the user wants to print out the estimation results:

```
SB_PrintEstimates(&bII)
```

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

Not clear yet. Possibly offer one function adapted to gretl's package GUI to test for the number of breaks, and another one to estimate the model with a prespecified number of breaks.

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.

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

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 `SB_Global` or `SB_Sequ`).
The number of breaks to be imposed needs to be passed as `n` only if the method "FIX" is chosen.
Returns scalar (just indicates success if 0). Its arguments are:

1. `b`: a model bundle (in pointer form) **(required)**
2. `method`: string to specify break determination method (one of "BIC", "LWZ", "SEQ", "REP", "FIX"; optional, if instead `b.method` exists)
3. `n`: integer to specify number of breaks for the "FIX" method (optional, except if `method==''FIX''`)

`SB_Global(bundle *b)`

Calls the procedure to obtain the break dates and the associated sum of squared residuals for all numbers of breaks between 1 and m. (Break dates in `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 `b.BICBreaks` and `b.LWZBreaks`.

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

1. `b`: a model bundle (in pointer form) **(required)**

`SB_PrintEstimates(bundle *b)`

Prints the current estimates, returns scalar (just indicates success if 0). Argument:

1. `b`: a model bundle (in pointer form) **(required)**

`SB_PrintOptions(bundle *b)`

Prints out the currently set options for the model contained in bundle `b`. Returns nothing. Arguments:

1. `b`: a model bundle (in pointer form) **(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", "Fixb", "PrintIter". **(required)**

Usage example:

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

`SB_Setup(series y, const list Z, const list X, int MB, scalar Epsilon)`

Returns a bundle to be passed around subsequently (in pointer form). 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. `MB`: scalar, maximum number of breaks

5. `Epsilon`: trimming parameter; it determines how many observations must exist before and after each break. (default: 0.1)

```
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.Eps1` (0.1; this is passed to SB_Setup via the "Epsilon" argument) – trimming parameter

- `b.MaxBreaks` (3; passed via the "MB" argument) – 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

- `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
- `b.p` – number of X elements
- `b.MaxIter` = 20 – internal iteration limit
- `b.Eps` = 0.00001 – <internal convergence criterion>

4.2 Matrices

- `b.SEQBreaks` (col vector with length 4, = number of considered signif. levels) – suggested numbers of breaks of the sequential procedure for each significance level
- `b.my` – dependent variable
- `b.mX` – regressors with non-breaking coefficients (may be empty)
- `b.mZ` – regressors with changing coefficients
- `b.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.SEQDates` – matrix for the break dates determined by the sequential procedure at various significance levels (signif. levels in rows)
- `b.REPDates` – similar matrix for the break dates determined by the repartition procedure
- `b.SupFDates` – vector for the break dates corresponding to the SupF tests
- `b.SupFTest` – vector of tests statistics for $l + 1$ against l breaks, for all possible l
- `b.Siglev` – hardcoded significance levels, corresponding to the CV functions that only contain the associated values
- `b.Coef` – coefficients of the estimated model
- `b.VCV` – covariance matrix of the estimated coefficients
- `b.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.BigMat` – <internal>
- `b.Betaini` – <internal>
- `b.global` – <internal>

4.3 Series

- `b.y` (Not needed internally because work is done on `b.my`. But the user could grab it from the bundle later.)

4.4 Strings

- `b.yname` – name of the dependent variable
- `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

- `b.Xnames` – names (one string per name) of the non-breaking variables
- `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:

$$y_t = \beta_0 + \beta_1 z_t + \beta_2 x_t + \varepsilon_t$$

where

$$\begin{aligned} \beta_0 &= 1 \text{ for } t \leq 150, -1 \text{ otherwise} && (1) \\ \beta_1 &= 1 \text{ for } t \leq 100, -1 \text{ otherwise} && (2) \\ \beta_2 &= 1 && (3) \\ V(\varepsilon_t) &= 1 && (4) \end{aligned}$$

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

# initialize
list Z = const z
list X = x
smpl full

bundle Model = SB_Setup(y, Z, X, 3, 0.2)
strings opts = defarray("HetvarYes")
```

```
SB_SetOptions(&Model, opts)

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)
```

A.2 Output of the example script

```
StrucBreak 0.21, 2018-01-09 (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        Eps1 = 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)
     3        222.57590        60 (1999:12), 115 (2004:07), 181 (2010:01)

===================================================================
   OUTPUT FROM THE APPLICATION OF INFORMATION CRITERIA
===================================================================
tmp (4 x 3)

      Breaks          BIC          lwz
      0.0000      0.84143      0.84979
      1.0000      0.50187      0.58750
      2.0000     0.065795      0.22885
```

```
      3.0000      0.13015      0.37080

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
--------------------------------------------------------
    supF(1|0)   supF(2|0)   supF(3|0)
     156.872     166.195     110.696

Critical values:
           supF(1|0)   supF(2|0)   supF(3|0)
10%               9.37        7.91        6.43
5%               10.98        8.98        7.13
2.5%              12.59       10.00        7.92
1%               14.92       11.30        8.95
--------------------------------------------------------
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.)
10%          196.87           10.46
5%          203.21           12.15
2.5%          209.24           13.87
1%          219.44           16.52
********************************************************
supF(l+1|l) tests using global optimizers under the null
--------------------------------------------------------
supF(2|1)         149.09        1999:12
supF(3|2)         1.05        2005:12
--------------------------------------------------------
Critical values:   10%    5%  2.5%    1%
       supF(2|1)  10.92 12.55 14.22 16.69
       supF(3|2)  11.90 13.46 15.39 17.41


===================================================================
   OUTPUT FROM THE SEQUENTIAL PROCEDURE
===================================================================
sig. lev.   breaks
The first break found is at: 181 (2010:01)
The next break found is at: 60 (1999:12)
    10.0%        2
The first break found is at: 181 (2010:01)
The next break found is at: 60 (1999:12)
     5.0%        2
The first break found is at: 181 (2010:01)
The next break found is at: 60 (1999:12)
     2.5%        2
The first break found is at: 181 (2010:01)
```

```
The next break found is at: 60 (1999:12)
     1.0%         2
********************************************************
Output from the repartition procedure for the 10.0% significance level
----------------------------------------
The updated break dates are :
60 (1999:12)        181 (2010:01)
********************************************************
Output from the repartition procedure for the 5.0% significance level
----------------------------------------
The updated break dates are :
60 (1999:12)        181 (2010:01)
********************************************************
Output from the repartition procedure for the 2.5% significance level
----------------------------------------
The updated break dates are :
60 (1999:12)        181 (2010:01)
********************************************************
Output from the repartition procedure for the 1.0% significance level
----------------------------------------
The updated break dates are :
60 (1999:12)        181 (2010:01)

===================================================================
   OUTPUT FROM ESTIMATION OF THE MODEL WITH 2 BREAKS (fixed)
===================================================================

Dependent variable: y
----------------------------------------------------------------
Unbreakables

             coefficient   std. error     z     p-value
  -----------------------------------------------------
  x            1.01879     0.0643043    15.84   1.57e-56 ***

--------------------------------------------------------------
Subsample 1: 1995:01 - 1999:12

             coefficient   std. error     z     p-value
  -----------------------------------------------------
  const       1.09134       0.130655    8.353   6.66e-17 ***
  z           0.927268      0.129479    7.162   7.98e-13 ***

Subsample 2: 2000:01 - 2010:01

             coefficient   std. error     z      p-value
  ------------------------------------------------------
  const        0.993910    0.0846317     11.74   7.59e-32 ***
  z           -1.01908     0.0899632    -11.33   9.57e-30 ***

Subsample 3: 2010:02 - 2014:12

             coefficient   std. error     z      p-value
  ------------------------------------------------------
  const       -0.973438     0.131084    -7.426   1.12e-13 ***
  z           -0.859305     0.135792    -6.328   2.48e-10 ***
```

```
--------------------------------------------------------------
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

- 0.21, 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