The ADMBP addon for gretl

Artur Tarassow

Version 0.99

Changelog

- Version 0.99 (March, 2021)
  - fix bug with the 'const' modifier
  - enforce decimal points for plotting
  - replace deprecated sprintf command by function
  - make use of r/cnameset()
  - minimum gretl version required is 2020a
- Version 0.96 (April, 2018)
  - fix bug with font size in irf_plotadv() function
- Version 0.95 (April, 2018)
  - add the stationary block-bootstrap as as another re-sampling option
- Version 0.9 (Sept., 2017)
  - Initial version

Contents

1 Introduction

The `ADMBP` (ARDL Dynamic Multiplier Bootstrap Package) is a collection of gretl scripts to estimate ARDL (autoregressive distributed lag) models and associated bootstrap coefficients as well as bootstrap dynamic multipliers.

This function package comprises the following features:

- Automatic lag selection of optimal ARDL model based on information criteria
- Estimation of unconditional and conditional ARDL models
- Bootstrap coefficient estimates using 4 different bootstrap procedures
- Computation of bootstrap dynamic multipliers
- Computation of bootstrap error-correction coefficients and bootstrap long-run dynamic multipliers under the assumption of cointegration
- Inference on long-run multipliers using either the bootstrap method or Bewley's IV approximation

In case you want to test for a long-run cointegrating relationship between the series of interest, I recommend to use the gretl user-written function package `cointARDL (LINK: here)` which comprises two bootstrap based tests using the single-equation cointegration framework. The help file of the `cointARDL` package can also be seen as a supplement to this help file.

Another user-written function package for gretl comprising facilities to compute dynamic multipliers was recently published by Oleh Komashko. His `lagreg` function package shares a few ideas with our package. Both packages provide useful tools.

2 The ARDL model

Begin with the ARDL(1,1) in levels model

$$y_t = \phi_1 y_{t-1} + c_0 x_t + c_1 x_{t-1} + u_t, \quad t = 1, ..., T$$

where ϕ_1, c_0 and c_1 are unknown parameters, and

$$x_t = x_{t-1} + e_t$$

where both u_t and e_t are some error terms.

The ARDL model can be re-written in the *error-correction* ECM(-ARDL) form:

$$\begin{aligned} \Delta y_t &= -(1-\phi_1)\left[y_{t-1} - \frac{c_0 + c_1}{1-\phi_1}\right] + c_0 \Delta x_t + u_t \\ &= \rho(y_{t-1} - \beta x_{t-1}) + c_0 \Delta x_t + u_t \end{aligned}$$

where $\rho = -(1-\phi_1)$ denotes the so called error-correction coefficient (speed of adjustment back to the long-run attractor). The model is dynamically stable as long as $|\rho| = |1-\phi_1| < 1$.

The general level ARDL(p,q) model (here without any deterministics) can be written as

$$y_t = \sum_{j=1}^{p} \phi_j y_{t-j} + \sum_{j=0}^{q} c_j x_{t-j} + u_t$$

where p and q denote the respective lag length. In case the lowest lag of the exogenous regressor is $q^{min} > 0$ a so called *unconditional* ARDL is estimated. If the contemporaneous value x_t is included, the so called *conditional* ARDL is specified (for details see Pesaran/Shin, 1999).

The following assumptions are made:

1. $u_t \sim iid(0, \sigma_u^2)$
2. e_t is a general linear stationary process with zero mean and finite variance.
3. u_t and e_t are uncorrelated for all lags such that x_t is strictly exogenous w.r.t. u_t
4. $|\phi_1| < 1$, so that the model is dynamically stable.

All model parameters are estimated by OLS using gretl's internal `ols` command.

2.1 Dynamic multipliers

The cumulative dynamic multiplier effects of x_t on y_t at horizon h are evaluated as

$$m_h = \sum_{j=0}^{h} \frac{\partial y_{t+h}}{\partial x_t}.$$

For $h \to \infty$, m_h converges to the **long-run multiplier** $\beta = -\frac{c_0+c_1}{\rho}$. In case the conditional ARDL is estimated, c_0 is the so called **impact multiplier**. Bootstrap confidence intervals are computed similarly to the algorithm explained in the following sub-section 2.2.

2.2 Inference on long-run multipliers

Using the ARDL framework, the long-run multiplier is a *non-linear* transformation. In the literature, the null hypothesis that the long-run multiplier $\hat{\beta} = 0$ is usually tested by means of the Delta method which is an approximation. Typically, however, the Delta method yields rather poor approximation. We offer two alternatives to compute the standard deviation of the long-run multipliers.

Bootstrap inference Using bootstrap sampling we can estimate the distribution of β. The algorithm works as follows:

1. Estimate the ARDL(p, q) model for fixed lags p and q by OLS

$$y_t = \sum_{j=1}^{p} \phi_j y_{t-j} + \sum_{j=0}^{q} c_j x_{t-j} + u_t$$

   and store all coefficients $\hat{\phi}_i$ and $\hat{c}_i$, and the estimated residuals $\hat{u}_t$ where the initial values are taken to be fixed in the statistical sense.

2. Construct the bootstrap sample, $\{y_t^*\}$, recursively from the first step with T randomly drawn bootstrap errors, u_t^*, generated using re-centered residuals, $\hat{u}_t^c := \hat{u}_t - T^{-1} \sum_{i=1}^{T} \hat{u}_t$. The same initial values are used for each generated series, $y^*_{-p+1,\ldots,}y^*_0 = y_{-p+1,\ldots,}y_0$.

3. Using the bootstrap sample, $\{y_t^*\}$, estimate the ARDL(p, q) model using OLS. Check for dynamic stability. If stability is ensured proceed with the next step, otherwise go back to step 2 and draw from another set of residuals.

4. Given the bootstrap parameter estimates, compute the long-run multiplier$\hat{\beta}^*$.

5. Repeat steps 1 to 4 B times.

6. Compute the standard deviation of all B bootstrap long-run multipliers $\hat{\beta}^*$.

Resampling is done by means of gretl's fast `resample()` function. In future versions of this package, we may make use of gretl's MPI toolbox to accelerate computations.

Bewley's IV approach Bewley proposed an alternative representation (see Bewley, 1979 and Hassler/Wolters, 2006) of the ARDL model according to which the long-run multipliers of vector β are just the coefficients of vector $\mathbf{x_t}$. His idea is based on an IV regression using lags of y_t and x_t as valid instruments. This estimation is automatically done by the `Bewley()` function.

For 2SLS estimates, we rely on gretl's native `tsls` function using the `--robust` option to compute HAC standard errors per default.

2.3 Bootstrap variants

The user can choose between 4 bootstrap methods. In the `ADMBP` package we generate for each replication an artificial data set of the same length as the original data set on the assumption that the estimated version of the core model is the true data-generating process, using the observed initial values of each variable, the estimated model, and a set of random innovations. These innovations can be obtained using 4 different ways. The *parametric* bootstrap draws from a normal distribution. The *non-parametric* bootstrap re-samples with replacement from the estimated residuals.

To illustrate the two semiparametric wild bootstrap approaches, start from the wild bootstrap DGP $y_t^* = X_t\tilde{\beta} + f(\tilde{u}_t)v_t^*$ where $\tilde{\beta}$ denotes the least squares estimates, and $f(\tilde{u}_t)$ is a transformation of the t^{th} residual $\tilde{u}_t$, and v_t^* is a random variable with mean 0 and variance 1.[1] The *wild uniform* bootstrap assumes that v_t^* is drawn from a uniform distribution. In contrast, the Rademacher two-point distribution is given by

$$v_t^* = \begin{cases} -1 & \text{with probability } 0.5 \\ 1 & \text{with probability } 0.5 \end{cases}$$

The stationary block-bootstrap was proposed by Politis/Romano (1994).[2] This bootstrap attempts to mimic time-dependency in the time-series. The mean block-size is determined by $b = round(1.75 \times T^{1/3})$ where T denotes the sample length.

`btype`	Bootstrap type
0	parametric
1	non-parametric
2	wild uniform
3	wild Rademacher
4	stationary block-bootstrap

[1] For further details, see e.g. russell-davidson.arts.mcgill.ca/e761/rd-jgm-bootstrap.pdf .

[2] The work was published as Dimitris N. Politis and Joseph P. Romano: The Stationary Bootstrap, *Journal of the American Statistical Association*, Vol. 89, No. 428 (Dec., 1994), pp. 1303-1313.

3 Install and load the package

Since the `ADMBP` package is user-contributed it is publicly available on the gretl server. The package must be downloaded once, and loaded into memory each time gretl is started.

```
# turn extra output off
set verbose off
# Download package (only once need)
install ADMBP
# Load the ADMBP package into memory
include ADMBP.gfn
# Get some initial help
help ADMBP
```

4 Examples

4.1 Determine the optimal lag length

A first step of the modeling stage comprises the determination of the optimal lag length of the approximate ARDL specification. The function `lagselect()` makes use of gretl's `var` command to determine the lag length using some standard information criteria. For the determination of the lag length, only variable y_t enters as an endogenous while all other variables are taken as exogenous. A simple example follows using the AWM-macroeconomic dataset comprising various series observed at a quarterly frequency over 28 years. The objective is to estimate a consumption equation where real log consumption expenditures, c_t, is supposed to be a linear but dynamic function of its own lags as well as the log of real income, inc_t, and log of real wealth, w_t.

For illustrative purposes we set the maximum lag length to $p = q = 4$. Note that at the moment, the `ADMBP` package does not allow for gappy lag structures and the same lag length applies to all variables — which is also the standard case in VAR modeling. In the example we use the AIC information criteria to determine the optimal lag length. However, the user has the choice of using the BIC or the HQ criteria alternatively.

Before we can call the public function `lagselect()`, we need to set up the main function `setARDL()` whose structure we will introduce in the following sub-chapter.

```
open AWM.gdt --quiet set seed 1234
# Transform series
series inc = ln(YER)
series c = ln(PCR)
series w = ln(WLN/PCD)
# Define lists of regressors
list xlist = inc w
list rxlist = const
list Lall = c xlist rxlist
# Get rid of missing observations
smpl Lall --no-missing

# Set ARDL parameters
scalar pqmax = 4                # max. lag length
scalar condARDL = 1      # 1=conditional ARDL, 0=unconditional
scalar infocrit = 1     # info.-crit. for lag determinat.: 1=AIC, 2=BIC, 3=HQC

# Bootstrap and confidence interval parameters
scalar horiz = 40
scalar cilevel = 0.1
scalar btype = 2
scalar bootrep = 1000
scalar shocktype = 1

#-------------------------
# Set up the base bundle
#-------------------------
bundle b = setARDL(c, xlist, rxlist, pqmax, \
infocrit, condARDL, shocktype, btype, bootrep, horiz, cilevel)

#-------------------------------------------
# Determine the optimal lag length (optional)
#-------------------------------------------
lagselect(&b)
```

We obtain the following output where one can see that the AIC criteria suggests an ARDL(3,3) model

```
*** Lag selection based on 80 obs. ***
--------------------------------------
         AIC          BIC          HQ
1      -8.1029     -7.9243     -8.0313
2      -8.0844     -7.8164     -7.9770
3      -8.2191     -7.8618     -8.0758
4      -8.1952     -7.7486     -8.0161
The selected information criteria suggests an ARDL(3,3) in levels.
```

Note two things here: First, initially we have send a couple of necessary information to `setARDL()`

in form of a bundle. This bundle b holds all necessary information. Second, using these bundle information, we have called the function `lagselect()`. Calling this `lagselect()` function is an optional case.

4.2 Bootstrap dynamic multipliers

Once we have set up the ARDL(p, q) model, we can proceed with the computation of the dynamic multipliers. For estimating the model initially by OLS and running the bootstrap algorithm, we call the function `runARDL()` which does all the job for us. Running this rather small ARDL model with 1000 bootstrap iterations takes only 0.7 sec. on a 5-year old i7 Laptop.

```
#---------------------------------------
# Compute bootstrap dynamic multipliers
#---------------------------------------
set stopwatch
runARDL(&b)
printf "This took %.3f sec.\n", $stopwatch

print b   # print information contained in the bundle
```

The corresponding gretl output reports the estimate using the non-bootstrap dataset.

```
*****************************************
*** ARDL (3) in levels regression       ***
*****************************************
Settings for dynamic multiplier computation
Shock type: permanent Bootstrap iterations: 1000
Bootstrap type: non-parametric
Confidence interval: 0.90
OLS, using observations 1977:4-1997:4 (T = 81)
Dependent variable: Y
             coefficient   std. error   t-ratio   p-value
-------------------------------------------------------
const       0.212495     0.0779382     2.726     0.0081    ***
Y_1          0.776561     0.108866      7.133     7.63e-10 ***
Y_2         0.180665     0.134492      1.343     0.1836
Y_3          0.196582     0.104365      1.884     0.0638    *
inc          0.643504     0.0796265     8.082     1.42e-11 ***
inc_1       0.525778     0.129367      4.064     0.0001    ***
inc_2        0.171476     0.142627      1.202     0.2334
inc_3       0.0836703    0.112801      0.7418    0.4608
w            0.357688     0.136087      2.628     0.0106    **
w_1         0.537362     0.185924      2.890     0.0051    ***
w_2          0.631706     0.185828      3.399     0.0011    ***
w_3         0.443685     0.122025      3.636     0.0005    ***

Mean dependent var   13.27836   S.D. dependent var   0.133614
Sum squared resid    0.000937   S.E. of regression   0.003685
R-squared            0.999344   Adjusted R-squared   0.999240
F(11, 69)            9557.579   P-value(F)           3.4e-105
Log-likelihood       345.4523   Akaike criterion     666.9046
Schwarz criterion    638.1712   Hannan-Quinn         655.3763
rho                  0.048209   Durbin's h           2.169477

Excluding the constant, p-value was highest for variable 141 (inc_3)
*** NOTE: 0.00 pct. of the 1000 bootstrap iterations failed due to instability.

This took 0.643 sec.
```

Note that all of the bootstrap models were dynamically stable. Remember that `runARDL()` can only be called one time after `setARDL()` as there is no reason to call it a second time, once all basic information are computed. Calling `runARDL()` again, requires to call `setARDL()` before.

4.3 Long-run multipliers and standard errors

Once the `runARDL()` function was called, we have generated all information to proceed. For computing the long-run dynamic multipliers, we just need to call the `LRbeta()` function.

```
#-------------------------------------------
# Compute bootstrap error-correction coeff. +
# long-run multipliers based on ARDL
#-------------------------------------------
LRbeta(&b)
# grab bootstrap mean, median and SD of long-run multiplier for variable "inc"
matrix LRinc = b.LRbeta_inc
```

The corresponding output is:

```
****************************************
*** ARDL (3) in levels regression       ***
****************************************
Settings for dynamic multiplier computation
Shock type: permanent
Bootstrap iterations: 1000
Bootstrap type: non-parametric
Confidence interval: 0.90
****************************************
*** Error-correction model bootstrap   ***
*** coefficient estimates              ***
****************************************
Error correction coeff. rho (bootstrap values)
-----------------------------------------------
        Median  Median   S.D.
rho -0.2896   -0.2815   0.0880

Long-run multipliers (bootstrap values)
-----------------------------------------------
        Mean    Median   S.D.
Y     2.7925   2.5527   1.2458
inc     0.9955   0.9907   0.1305
w     0.0314   0.0335   0.1104

Bewley's long-run multipliers (2SLS point estimates)
-----------------------------------------------
        Coeff.  S.D.
inc     0.9904   0.1641
w     0.0402   0.1361
****************************************
```

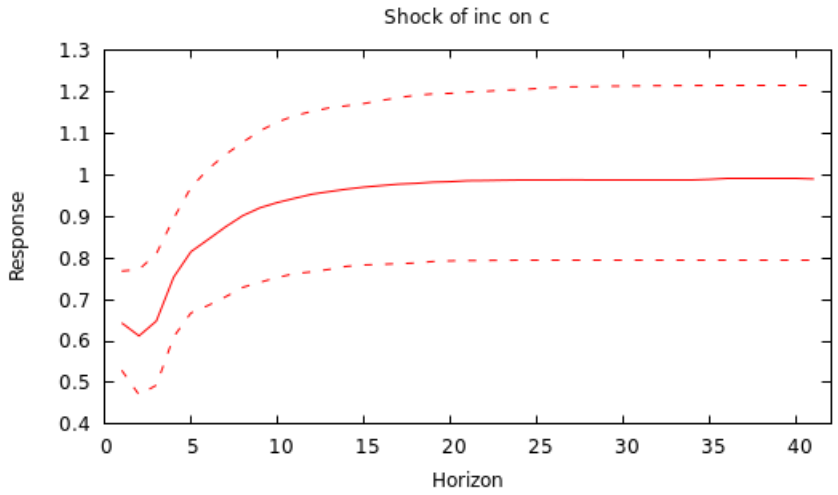
One can see that the bootstrap yields somehow smaller standard deviations compared to Bewley's approach. However, the long-run multipliers are of similar magnitude for *inc* and w.

4.4 Plot the dynamic multipliers

We can easily plot the associated dynamic multipliers for each variable over some horizon using the `irf_plot()` function. The following listing compiles a figure with the corresponding output.

```
# Plot the response of "c" to a permanent unit increase in "inc"
irf_plot(b.DM_inc, "Shock of inc on c", "Horizon", "Response")
```

Here is the corresponding output with 90% confidence intervals.



Alternatively we also offer a more sophisticated plotting function which includes more plotting options for the user, e.g. setting font size, line width, title and axis labeling. In the following, we provide a loop function which computes the IRFs for all possible cases and stores the three figures as pdf files, respectively.

```
list Lplot = Lall - rxlist
strings S = varnames(Lplot)
S[1] = "Y" # The endogenous is always named "Y"

#-------------------------------------------------------
# Plot multipliers with additional options for drawing
# Save as *.pdf
#-------------------------------------------------------
scalar font = 26  # set font size
scalar lw = 2.0   # set line width
loop foreach i Lplot -q
        string figpath = "@dotdir/irf_$i.pdf"
        string title = "foo"
        string xlab = "put a x label here"
        string ylab = "put a y label here"
        string shock = S[i]     # read out the name of the shock
        irf_plotadv(b.DM_@shock, figpath, title, xlab,ylab, \
        font, poslegend, lw)
endloop
```

5 List of public functions

5.1 setARDL

```
setARDL(series Y, list xlist, list rxlist[null], int pqmax[1::4], int crit[1:3:1],
bool condARDL[1], int shocktype[0:1:], int btype[0:3:1], int bootrep[99::999],
int horiz[1::12], scalar cilevel[0.01:0.99:0.1], bool verb[1], scalar
failstop[0.01:0.9:0.2])
```

Return type: `bundle`

The function arguments are:

1. `series Y`: dependent variable
2. `list xlist`: list of exogenous regressors
3. `list rxlist[null]`: list of additional deterministic variables (optional)
4. `int pqmax[1::4]`: Max. lag length in levels for y and xlist variables (default 4)
5. `int crit[1:3:1]`: Information criteria for automatic lag selection, 1=AIC, 2=BIC, 3=HQC (default 1)
6. `bool condARDL[1]`: include contemporaneous values of xlist (default yes)
7. `int shocktype[0:1:]`: Compute dynamic multiplier for transitory (=0) or permanent (=1) change in xlist-variable (default 1).
8. `int btype[0:4:1]`: Select bootstrap type, 0=parametric, 1=non-parametric, 2=wild (uniform), 3=wild (Rademacher), 4=stationary block, default 1.
9. `int bootrep[99::999]`: Number of bootstrap iterations (default 999)
10. `int horiz[1::12]`: Number of horizons to compute dynamic multipliers (default 12)
11. `scalar cilevel[0.01:0.99:0.1]`: Width of confidence interval, alpha=1-cilevel (default 0.1 –> 90%)
12. `bool verb[1]`: Print details (default yes)
13. `scalar failstop[0.01, 0.9, 0.2]`: Fraction of failed bootstrap iterations before full stop (default 0.2)

This is the basic function for setting up all relevant series, parameter and options. This function always has to be called first.

5.2 lagselect

```
lagselect(bundle *b)
```

Return type: `void`

This function can be called *after* you have called `setARDL()`. Selects the optimal number of lags of an unrestricted ARDL(p, q) model by minimizing some chosen information criteria. The maximal lag is set by the scalar `pqmax`.

This corresponding scalar will be automatically updated in the relevant bundle once `lagselect()` as called. At the moment, the number of optimal lags has to be equal among all regressors of Y and *xlist*. Currently, the user cannot define a gappy lag structure.

5.3 ECterm

```
ECterm(bundle *b)
```

Return type: `series`

This function computes the error-correction term *aka* the cointegrating relationship, ζ_t, using the respective bootstrap median long-run coefficients. To be more concrete, assume that variables y_t and x_t are both I(1) variables and cointegrated where β refers to the long-run coefficient such that the error, ζ_t, is also stationary:

$$\zeta_t = y_t - \beta x_t.$$

The literature discusses the following five cases

$$\begin{aligned}
\textbf{(I)}\ &\text{no constant: } \zeta_t = y_t - \beta x_t \\
\textbf{(II)}\ &\text{restr. const.+no trend : } \zeta_t = y_t - \beta x_t - \delta_0/\rho \\
\textbf{(III)}\ &\text{unrestr. const+no trend: } \zeta_t = y_t - \beta x_t \\
\textbf{(IV)}\ &\text{unrestr. const +restr. trend: } \zeta_t = y_t - \beta x_t - \delta_1/\rho \\
\textbf{(V)}\ &\text{unrestr. const.+unrestr. trend: } \zeta_t = y_t - \beta x_t
\end{aligned}$$

where δ_0, δ_1 and ρ refer to the intercept, the coefficient of the linear trend and the error-correction coefficient. Currently we support the computation of all cases except case 4.

Based on our example where we estimated an ARDL model including an intercept term, we can compute the two respective error-correction term for the cases (II) and (III), respectively:

```
#-----------------------------------------------
# Compute bootstrap-based error-correction term
#-----------------------------------------------
ECterm(&b)
series ECM = b.ECM # error-correction term (case III)
ECMrc = b.ECMrc # error-correction term (case II)
gnuplot ECM ECMrc --time-series --with-lines --output=display
```

5.4 irf_plot

```
irf_plot(matrix m, string title, string xlab, string ylab)
```

Return type: `void`

The function arguments are:

1. `matrix m`: matrix with either one column (irf only), or with three columns (irf and confidence bounds).
2. `string title`: provide a string for the title.
3. `string xlab`: provide a string for the x-label.
4. `string ylab`: provide a string for the y-label.

This function plots the dynamic multipliers immediately on the screen.

5.5 irf_plotadv

```
irf_plot(matrix m, string fname, string title, string xlab, string ylab, int
fontsize[5::14], scalar lw)
```

Return type: `void`

1. `matrix m`: matrix with either one column (irf only), or with three columns (irf and confidence bounds).
2. `string fname`: provide the path and filename (+file extension, e.g. pdf, png, eps) for storing the graph.
3. `string title`: provide a string for the title.
4. `string xlab`: provide a string for the x-label.
5. `string ylab`: provide a string for the y-label.
6. `int fontsize[5::14]`: set the font size (default 14)
7. `scalar lw`: set the line width.

References

Bewley, R.A. (1979): The direction estimation of the equilibrium response in a linear dynamic model, *Economic Letters*, 3, 357-361.

Hassler, U. and J. Wolters (2006): Autoregressive Distributed Lag Models and Cointegration, *Allgemeines Statistisches Archiv* 0, 0–15.

Pesaran M.H. and Y. Shin (1999): An autoregressive distributed lag modelling approach to cointegration analysis, *Econometrics and Economic Theory in the 20th Century: The Ragnar Frisch Centennial Symposium*, Strom S (ed.). Cambridge University Press: Cambridge.