A Simple Bootstrap Package for MATLAB

Alex Carrasco Martinez
(alex.carmar93@gmail.com)

First year master in economics - PUC Rio

December, 2017

Abstract

The present document shows how to apply different bootstrap techniques to different data structure using MATLAB. It is worth making some general observations about the utility of these functions: a) they deal with multivariate data, b) some of them necessarily needs specific data arrangement, the following examples show the way we can construct these objects, c) the package includes a function to compute kernel density and cumulative distribution estimators (mykernel.m) for *multivariate data* which is extremely useful in non-parametric smoothed bootstrap, d) bootstrapping is performed with a reasonable low computational cost, e) each function has its own documentation, and f) I have tried to test the codes in several ways and can be applied to a wide range of econometric models, however, there could be some bugs that I have not found yet so if you find some, please contact me.

This document does not have any relation with the university beyond the fact that I study there. Errors are my own.

Contents

1 Introduction

Bootstrap is a method to estimate the distribution of an estimator or test statistic by resampling one's data. It consists to treating the sample as if it was the population in order to evaluate the distribution of interest. In its more general form, we have a sample of size N and we want to estimate a parameter or determine the standard error or confidence interval about the parameter. If we do not make any parametric assumptions, we may find this difficult to compute. Then bootstrap provides a way to this. A general bootstrap algorithm is as follows:

Algorithm 1. General bootstrap algorithm. *Given a sample $\mathbf{x}_1, \dots, \mathbf{x}_N$ where $\mathbf{x}_i \in \mathbb{R}^k$ for $i = 1, \dots, N$, then for $b = 1$ to B:*

1. *Draw a bootstrap sample of size N using a method given in the following and denote this new sample as $\mathbf{x}_1^{(b)}, \dots, \mathbf{x}_N^{(b)}$.*
2. *Compute the statistic of interest $T_N^{(b)}$ using $\mathbf{x}_1^{(b)}, \dots, \mathbf{x}_N^{(b)}$, i.e., $T_n^{(b)} \equiv T_n(\mathbf{x}_1^{(b)}, \dots, \mathbf{x}_N^{(b)})$.*

Use these B bootstrap replications to obtain bootstrapped version of the statistic.

The rest of the document exposes the package dividing the methods by the type of data they are supposed to deal with: section 2 presents methods for dealing with independent and identically distributed or serially uncorrelated samples, while in section 3, functions to deal with dependent data are shown.

2 Random sample

Along this section I assume that $\mathbf{X}_1, \ldots, \mathbf{X}_N$ are independent and identically distributed according to an unknown distribution function $F_0(\mathbf{x}) = \mathbb{P}(\mathbf{X} \leq \mathbf{x})$. Then the exact finite-sample distribution of T_N is $G_N = G_N(t, F_N) = \mathbb{P}(T_N \leq t)$. The problem is to find a good approximation to G_N. Bootstrap methods imply to substitute F_0 by a consistent estimator F_N of F_0. Mainly, we can use the empirical distribution function (EDF, hereafter) and kernel density estimations because these estimators are consistent under mild regularity conditions. Based on Chernick (1999), Horowitz (2001), Cameron and Trivedi (2005), Chernick and LaBudde (2011), and Cameron and Miller (2015).

2.1 Naive Bootstrap: `naive_boots.m`

The function script `naive_boots.m` implements the following algorithm:

```
***Algorithm 2. Naive bootstrap algorithm.*** *Given a sample* $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ *where* $\mathbf{x}_i \in \mathbb{R}^k$ *for* $i = 1, \ldots, N$, *do:*
  1. Compute the EDF of $\mathbf{X}$, $\hat{F}_N(.)$.
     Then for $b = 1$ to $B$:
  2. Draw a bootstrap sample of size $N$ from $\hat{F}_N(.)$ and denote this new sample as $\mathbf{x}_1^{(b)}, \ldots, \mathbf{x}_N^{(b)}$. That is equivalent to extract a sample with replacement from $\mathbf{X}$.
  3. Compute the statistic of interest using this new sample, i.e., $T_n^{(b)} \equiv T_n(\mathbf{x}_1^{(b)}, \ldots, \mathbf{x}_N^{(b)})$.
  4. $b = b + 1$.
Use these $B$ bootstrap replications to obtain bootstrapped version of the statistic.
```

The syntax and documentation of this function is:

```
[t,Y,F,Xgrid]=naive_boots(X,T,varargin)

% Obtains bootstrap sample for any statistic of interest using naive
% bootstrap method.
%
% ==============
%  Syntax:
% ==============
%       [t,Y,F,Xgrid]=naive_boots(X,T)
%       [t,Y,F,Xgrid]=naive_boots(X,T,...)
%
% ==============
%   Inputs
% ==============
%  X:   is the observed variables matrix (size: nobs x m)
%  T:   is a handle function for statistic computation.
%
%  **********
%    Options
%  **********
%  [1] 'n_replic': scalar that sets the number of bootstrap samples. [
   default: B=100]
```

```
%  [2] 'n_grid'  : scalar that sets the number of grid in the
   computation of the empirical distribution function. [default: n_grids
   =50]
%  [3] 'size'    : scalar that sets the size of each bootstrap sample.
   [default: N= number of observations in X]
%
% ==============
%  Outputs
% ==============
%  t      : bootstrap sample for the statistic of interest.
%  Y      : bootstrap sample of observed data (based on X).
%  F      : empirical distribution function (EDF).
%  Xgrid : Grid used in EDF computation.
% ============================================================
```

Example 1. **(Estimating bias)**. Let be $X = \{x_i\}_{i=1}^{150}$ a random sample from a Poisson distribution function ($\lambda = 10$). Since $\mathbb{E}(X_i) = \lambda$ and $\mathbb{V}(X_i) = \lambda$, then we have three alternatives to estimate λ:

$$\hat{\lambda}_1 \equiv \overline{x} = \frac{\sum_{i=1}^{N} x_i}{N} \qquad \text{[unbiased]} \tag{2.1}$$

$$\hat{\lambda}_2 \equiv s_1^2 = \frac{\sum_{i=1}^{N} (x_i - \overline{x})^2}{N-1} \tag{2.2}$$

$$\hat{\lambda}_3 \equiv s_2^2 = \frac{N-1}{N} s_1^2 \tag{2.3}$$

The bias of the statistic T_N is defined as $\mathsf{bias}(T_N) = \mathbb{E}(T_N) - T_0$ where T_0 represent the parameter of interest. Thus, based on the bootstrap sample $\{T_N^{(b)} | b = 1, \ldots, B\}$ the bias can be estimated as

$$\frac{1}{B}\sum_{b=1}^{B} T_N^{(b)} - T_N \tag{2.4}$$

Hence, the bias adjusted estimator of λ is

$$\hat{\lambda}_r - \mathsf{bias}(\hat{\lambda}_r) = 2\hat{\lambda}_r - \frac{1}{B}\sum_{b=1}^{B} \hat{\lambda}_r^{(b)} \tag{2.5}$$

for $r = 1, 2, 3$. To obtain this adjusted estimators we can implement the function naive_boots.m as follows:

```
%% [I] Naive bootstrap
rng(1115);       % controlling random numbers generator (for reproducibility)
N      = 150;
B      = 3000;
grids = 100;

lambda=10;
T=@(x) [mean(x) var(x) var(x,1)];  % estimators

X=random('poisson',lambda,N,1);
[t,Y,EDF,Xgrid]=naive_boots(X,T,'n_replic',B,'n_grid',grids);

% Printing results
fprintf('====================================\n');
fprintf('Naive Bootstrap: estimating bias\n');
fprintf('====================================\n');
lambest = T(X)
```

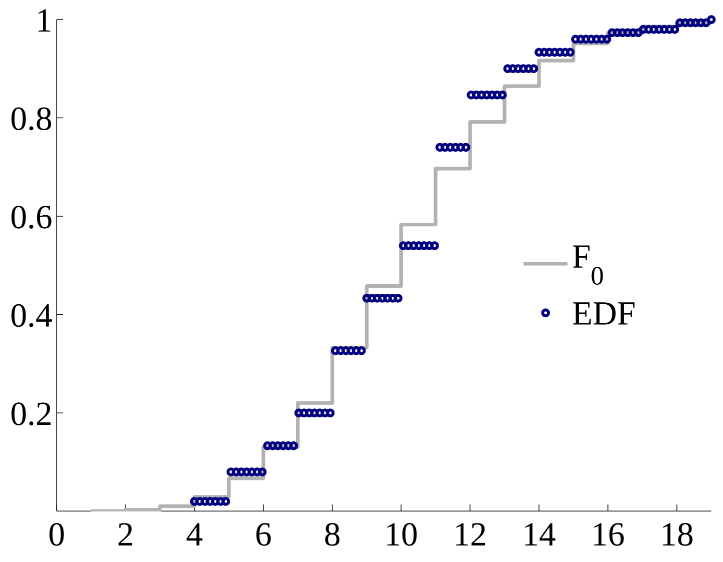
```
bias     = mean(t) - T(X)
badj     = T(X) - bias
```

Running this section we obtain estimation for λ:

```
======================================
Naive Bootstrap: estimating bias
======================================

lambest =

    9.9400    8.8487    8.7897


bias =

    0.0185   -0.0263   -0.0261


badj =

    9.9215    8.8750    8.8158
```

Additionally, as you can see, 10th line of the code also ask for the EDF and it is shown at Fig. 1.

FIGURE 1. *Empirical distribution function versus true distribution*



2.2 NON-PARAMETRIC BOOTSTRAP: `nonparam_boots.m`

For statistic as percentiles, naive bootstrap will not lead to reasonable estimations, then we would like to smoothed the estimation of $\hat{F}_N(.)$. Hence, kernel estimators becomes important part of smoothed bootstrapping.

Algorithm 3. Non-parametric bootstrap algorithm. *Given a sample* $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ *where* $\mathbf{x}_i \in \mathbb{R}^k$ *for* $i = 1, \ldots, N$ *with* q *continuous variables,* $\mathbf{x}^c$*, and* r *discrete variables,* $\mathbf{x}^d$*, do:*

1. *Compute the kernel distribution estimator of* $\mathbf{X}$, $\hat{F}_N(.)$:

$$\hat{F}_N(\mathbf{x}^c, \mathbf{x}^d) = N^{-1}\sum_{i=1}^{N}\Big(\prod_{j=1}^{q} G[(x_j^c - X_{i,j}^c)/h_j]\Big)\Big(\sum_{u\leq \mathbf{x}^d} L(\mathbf{x}_i^d, u)\Big) \tag{2.6}$$

where $G(\psi) = \int_{-\infty}^{\psi} k(v)dv$, $L(\mathbf{x}_i^d, u) = \prod_{s=1}^{r} l(X_{i,s}^d, u)$, $k(v)$ *is a kernel function, and* $l(X_{i,s}^d, u) = N^{-1}\sum_{i=1}^{N}\mathbb{I}(X_{i,s}^d = u)$.

Then for $b = 1$ *to* B:

2. *Draw a bootstrap sample of size* N *from* $\hat{F}_N(.)$ *and denote this new sample as* $\mathbf{x}_1^{(b)}, \ldots, \mathbf{x}_N^{(b)}$. *That is equivalent to extract a sample with replacement from* $\mathbf{X}$.
3. *Compute the statistic of interest using this new sample, i.e.,* $T_n^{(b)} \equiv T_n(\mathbf{x}_1^{(b)}, \ldots, \mathbf{x}_N^{(b)})$.
4. $b = b + 1$.

Use these B *bootstrap replications to obtain bootstrapped version of the statistic.*

First of all, we need a function that computes kernel estimators. mykernel.m computes kernel density estimator and kernel distribution function estimator (KCE) for multivariate data and here is a breve documentation:

```
[f,F,Xgrid]=mykernel(X,varargin)

% Computes kernel density estimator (KDE) and kernel cumulative estimator (
   KCE).
% ==============
%  Syntax:
% ==============
%       [f,F,Xgrid]=mykernel(X)
%       [f,F,Xgrid]=mykernel(X,...)
%
% ==============
%   Inputs
% ==============
%  X:   is the observed variables matrix (size: nobs x m)
%  *********
%   Options
%  *********
%  [1] 'kernel'  : scalar that sets the kernel function used in the kernel
   distribution estimator:
%            [1] kernel=1, Uniform density
%            [2] kernel=2, Gaussian density
%            [2] kernel=3, Epanechnikov kernel function
%            [4] kernel=4, Triangular kernel function
%        [default: kernel=1].
%  [2] 'n_grid'  : scalar that sets the number of grid in the computation
   empirical distribution function. [default: n_grids=50].
%  [3] 'bandwith': sets the bandwith size. [Default: use Scott's rule of
   thumb: h=(4/((k+2)*N))^(1/(k+4))*std(X)]
%  [4] 'eachseries': Compute KDE for each series in database.
%
% ==============
%  Outputs
% ==============
%  f     : Kernel density estimator (KDE).
%  F     : Kernel distribution function (KCE).
%  Xgrid : Grid used in KCE computation.
%
```

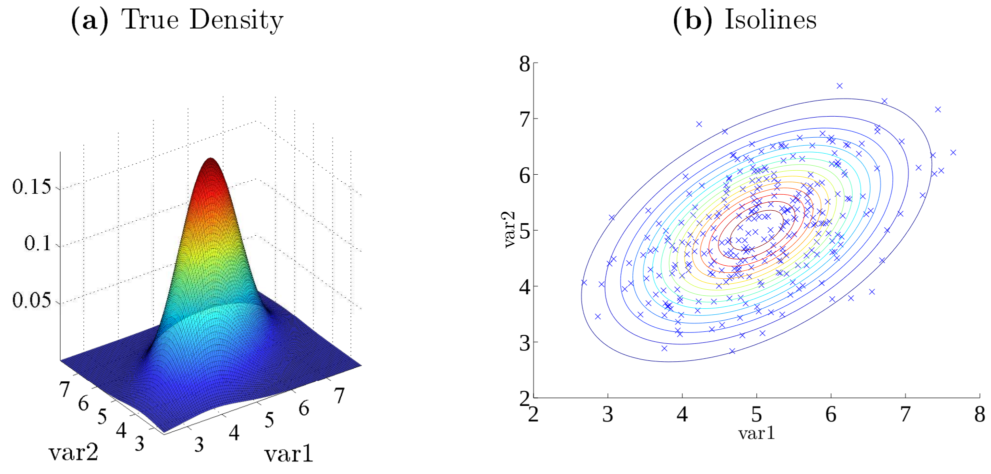
```
%============================================================
```

Example 2. **Bivariate unimodal Gaussian**. The objective of this example is to inspect two important results in non-parametric theory. I generate $N = 500$ draws from a bivariate distribution and study the performance of uniform, Epanechnikov and kernel estimator.

```
%% [II] Kernel densities
rng(1115);          % controlling random numbers generator (for
    reproducibility)
N  = 300;
S  = [1 .5; .5 1];    % covariance matrix
mu = 5*ones(2,1);     % mean
D  = mvnrnd(mu,S,N); % generating sample

% Uniform kernel estimator (default)
[fu,Fu,Xgrid] = mykernel(D,'n_grids',100);

% Gaussian kernel estimator (kernel=2)
[fn,Fn]  =  mykernel(D,'n_grids',100,'kernel',2);

% Epanechnikov kernel estimator (kernel=3)
[fe,Fe]  =  mykernel(D,'n_grids',100,'kernel',3);
```

Figure 2 plots the true density function and compared it with observed sample. This population density distribution can be contrasted with Fig. 3 (top panel) at which kernel density estimators are obtained. Both figures figure out a well known result in kernel density estimators: **precision decrease as more continuous variables are considered**. Other important result in non-parametric theory is that **more continuous variables does not reduce KCE's precision** as bottom panel of Figure 3 shows.

FIGURE 2. *Population distribution*

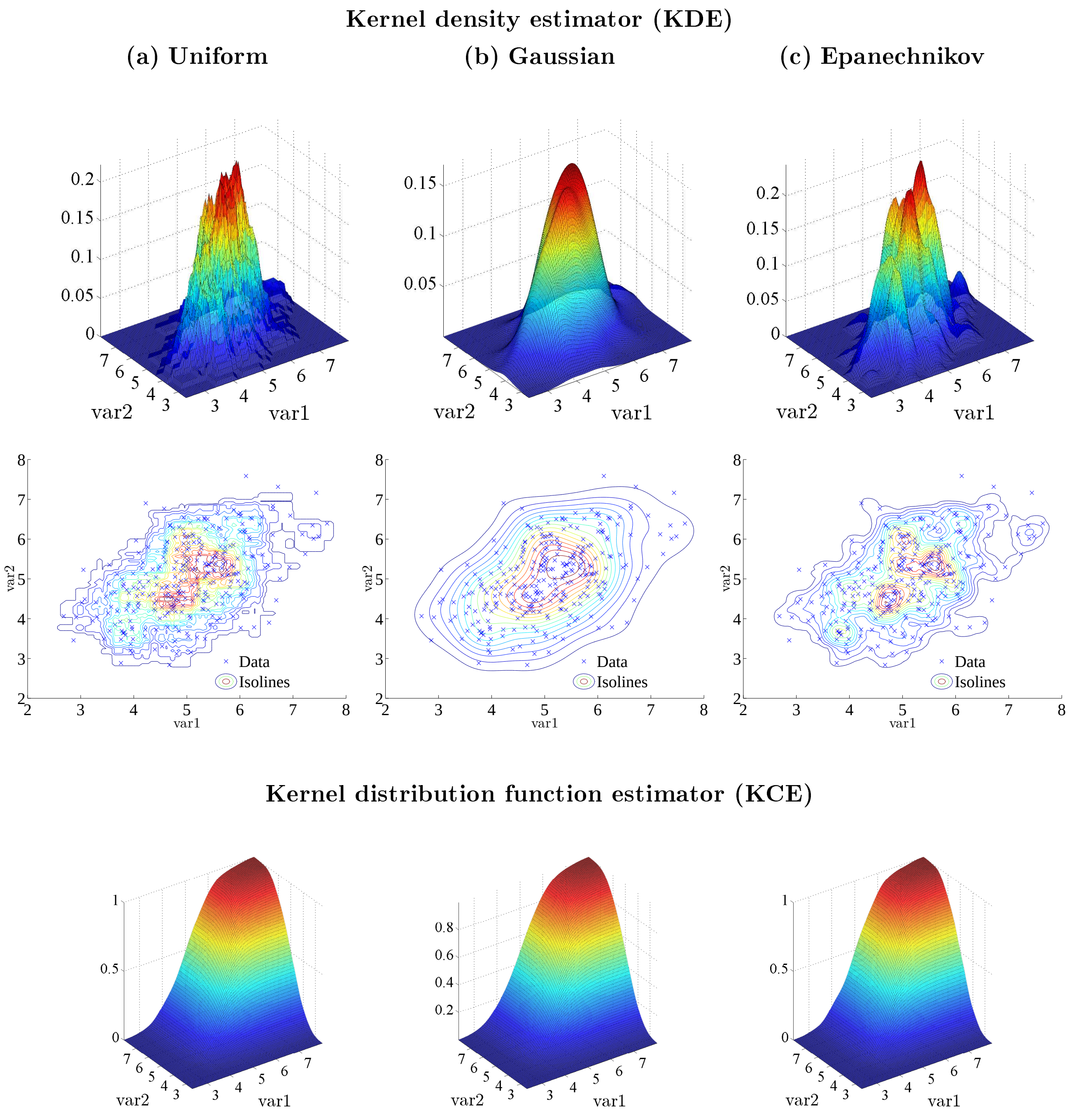


Once the KCE is obtained we must generate a random sample from this distribution. In general, I will follow:

Algorithm 4. ***(Generating random sample)****. Given $\hat{F}_N(x)$ evaluated on the grid X, for $i = 1$ to N,*

1. *Generate a draw from $U[0, 1]$, u.*
2. *Find x_i such that $\hat{F}_N(x_i) \approx u$.*

FIGURE 3. *Kernel function estimators*



At the end of this loop we have the random sample: $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

To used non-parametric sampling use the function:

```
[t,Y,F,Xgrid,f]=nonparam_boots(X,T,varargin)

% Obtains bootstrap sample for any statistic of interes using
% non-parametric smoothed bootstrap technique.
% ==============
%  Syntax:
% ==============
%       [t,Y,F,Xgrid]=naive_boots(X,T)
%       [t,Y,F,Xgrid]=naive_boots(X,T,...)
%
% ==============
%   Inputs
% ==============
%  X:    is the observed variables matrix (size: nobs x m)
%  T:    is a handle function for statistic computation.
%
%   **********
%     Options
%   **********
%   [1] 'n_replic': scalar that sets the number of bootstrap samples. [
    default: B=100]
%   [2] 'n_grid'  : scalar that sets the number of grid in the computation
    empirical distribution function. [default: n_grids=50]
%   [3]  'kernel'  : scalar that sets the kernel function used in the kernel
    distribution estimator. [default: Uniform pdf]. See 'mykernel.m'
    documentation.
%
% ==============
%  Outputs
% ==============
%  t      : bootstrap sample for the statistic of interes.
%  Y      : bootstrap sample of observed data (based on X).
%  F      : Kernel distribution function (KCE).
%  Xgrid  : Grid used in KCE computation.
%  f      : Kernel density estimator (KDE).
%============================================================
```

Example 3. (**IBOVESPA Perncetile's estimation**). In this example I study some statistics of the distribution of IBOVESPA series from January 2002 to October 2017. Figure 4 plots the historical series [panel (a)] and the kernel density estimator [panel (b)] comparing with a normal distribution [$\mathcal{N} \sim (0, 6)$]. To use non parametric bootstrap simply follow the lines:

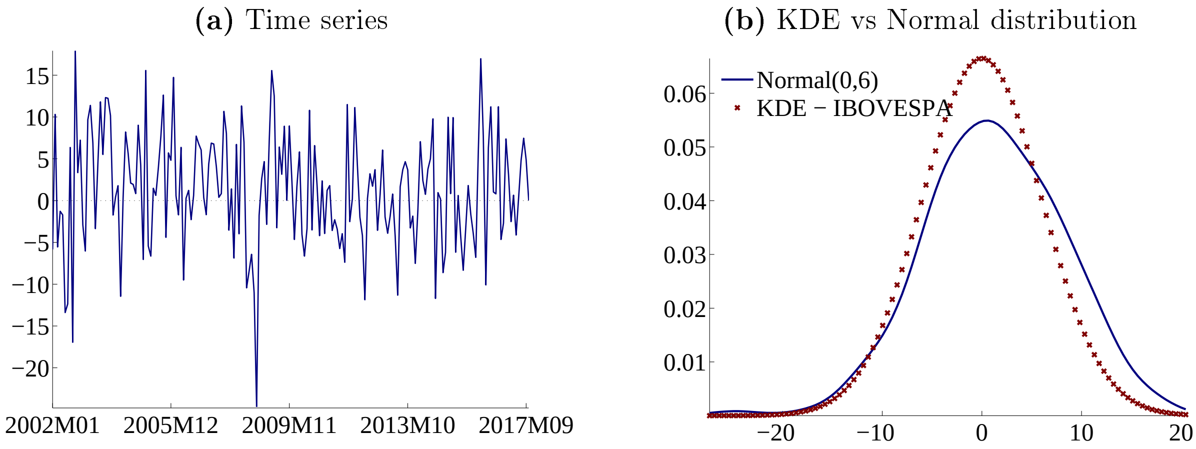
```
%% [III] Non-parametric bootstrap
load BOV_data.mat;        % IBOVESPA series: Dsp (series), ran (sample range)
p = [.05 .3 .5 .8 .95];   % percentiles of interest
T = @(x) [mean(x) quantile(x,p)];

% smoothed bootstrap using Gaussian kernel
[t,Y,F,Xgrid,f]=nonparam_boots(Dsp,T,'kernel',2,'n_replic',1e3);

% Printing results
fprintf('============================================================\n');
fprintf('Non-parametric Bootstrap: estimating percentiles (IBOVESPA)\n');
fprintf('============================================================\n');
fprintf('Estimation using sample\n');
T(Dsp)
fprintf('Estimation using bootstrap sample\n');
mean(t)
fprintf('Estimation using bootstrap sample (adjusted)\n');
2*T(Dsp)-mean(t)
```

Then given the matrix `t` we can compute biased adjusted estimators for the percentiles for IBOVESPA, see Table 2.

2.3 Residual Bootstrap: `residual_boots.m`

There are to basic approaches to bootstrapping in regression models and both can be applied to nonlinear as well as linear models. The first method is bootstrapping residuals. In theory, if the model is "nearly"

FIGURE 4. *IBOSPEVA from* 2002M1 *to* 2017M10



Source: Central bank of Brazil.

TABLE 1. *Percentile's estimations*

Estimator	**Mean**	**Percentile:** p =				
		5%	30%	50%	80%	95%
Sample estimator	1.1921	-10.4400	-2.6150	0.8400	6.9250	11.8100
Bootstrap estimator	0.9491	-10.8752	-2.8007	0.9131	7.1935	12.6883
Bootstrap estimator (adj.)	1.4350	-10.0048	-2.4293	0.7669	6.6565	10.9317

correct, the model residuals would be independent and identically distributed. Then, it is possible to use naive bootstrap to get a pseudosample, and then fit the model to the pseudosample and repeat.

`residual_boots.m` implements the residual bootstrap for linear models:

$$\mathbf{y}_i = \mathbf{X}_i\boldsymbol{\beta} + \mathbf{v}_i, \quad \forall\, i = 1, \dots, N \tag{2.7}$$

where $\mathbf{y}_i$ is a $(m \times 1)$ random vector, $\mathbf{X}_i$ is a $(m \times k)$ random matrix, $\boldsymbol{\beta}$ is the $(k \times 1)$ parameter vector, and $\mathbf{v}_i$ is a random vector with $\mathbb{E}[\mathbf{v}_i\mathbf{v}_j'|\mathbf{X}] = \Sigma$ for $i = j$ and zero otherwise. The function uses OLS estimators, however, in general, the residual bootstrap algorithm is:

Algorithm 5. Residual bootstrap algorithm. *Consider a linear model in the form of (2.7). Assume the we are using an estimator of $\boldsymbol{\beta}$, $T_N(.)$. Do:*

1. *Using $\hat{\boldsymbol{\beta}} = T_N((\mathbf{y}_1, \mathbf{X}_1), \dots, (\mathbf{y}_N, \mathbf{X}_N))$, estimate the EDF, $\hat{F}_N(.)$, of $\hat{\mathbf{v}}_i = \mathbf{y}_i - \mathbf{X}_i\hat{\boldsymbol{\beta}}$ for $i = 1, \dots, N$.*

   Then, for $b = 1$ to B:
2. *Get a sample from $\hat{F}_N(.)$ and call it $\{\mathbf{v}_1^{(b)}, \dots, \mathbf{v}_N^{(b)}\}$,*
3. *Construct the bootstrap sample: $\mathbf{y}_i^{(b)} = \mathbf{X}_i\hat{\boldsymbol{\beta}} + \mathbf{v}_i^{(b)}$ for $i = 1, \dots, N$*
4. *Compute and save, $\hat{\boldsymbol{\beta}}^{(b)} = T_N((\mathbf{y}_1^{(b)}, \mathbf{X}_1), \dots, (\mathbf{y}_N^{(b)}, \mathbf{X}_N))$,*
5. *$b = b + 1$.*

Use these B bootstrap replications to obtain bootstrapped version of the statistic.

The syntax is

```
[beta,St,V,bhat,S,D]=residual_boots(y,X,nobs,varargin)

% Obtains parameter estimator resamples using residual bootstrap method
% in a linear model:
%
%    y_i = X_i*beta + v_i
%
% where y_i is a (m x 1) random vector, X_i is (m x k), v_i is (m x 1),
% and beta is the (k x 1) parameter vector.
% ==============
%  Syntax:
% ==============
%      [beta,St,V,bhat,S,D]=residual_boots(y,X,nobs)
%      [beta,St,V,bhat,S,D]=residual_boots(y,X,nobs,...)
%
% ==============
%   Inputs
% ==============
%  y:   is a stack vector of y_i for i=1 to nobs (size: m.nobs x 1 )
%  X:   is a stack vector of X_i for i=1 to nobs (size: m.nobs x k )
%  nobs: is the number of observations in sample.
%   **********
%    Options
%   **********
%   [1] 'n_replic': scalar that sets the number of bootstrap samples (B).
%                   [default: B=100].
%   [2] 'size'    : scalar that sets the size of each bootstrap sample.
%                   [default: N=nobs].
% ==============
%  Outputs
% ==============
%  beta: (B x k) matrix with vector parameter estimations using bootstrap
   sample.
%  St  : (m x m x B) object with variance estimations for residuals.
%  V   : (nobs x m x B) object with residual bootstrap sample.
%  bhat: OLS estimation using the given sample.
%  S   : residual variance estimation using the given sample.
%  D   : residuals estimation using OLS.
%============================================================
```

Example 4. (**Panel data: OLS and GLS estimator**). The objective of this example is showing in a simply way how to use `residual_boots.m` function. First, I create three random samples: two of them from exponential distribution and the another one from a bivariate normal distribution. I construct the observed sample for $\mathbf{y}_i$ and, finally I study some properties of distribution for OLS and GLS estimators based on bootstrap sample. Hence, basically I construct a panel data and compare POLS and RE estimator. Using (2.7) notation:

$$\mathbf{X}_i = \begin{bmatrix} \mathbf{x}'_{i1} \\ \mathbf{x}'_{i2} \end{bmatrix} = \begin{bmatrix} x_{i,11} & x_{i,12} \\ x_{i,21} & x_{i,22} \end{bmatrix}, \quad \mathbf{v}_i = \begin{bmatrix} v_{i1} \\ v_{i2} \end{bmatrix}$$

$$\mathbf{y}_i = \begin{bmatrix} y_{i1} \\ y_{i2} \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$$

where $\mathbf{x}_{i,1j} \sim \exp(\lambda_1)$, $\mathbf{x}_{i,2j} \sim \exp(\lambda_2)$, and $\mathbf{v}_i|\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ for $j = 1, 2$ and $i = 1, \ldots, N$.

```
%% [IV] Residual bootstrap (Panel data)
rng(1115);    % controlling random numbers generator (for reproducibility)
N = 500;
B = 200;
lambda = [0.1  0.5];    % parameters for each covariate
```

```
Sigma   = 15*[1 .5; .5 1]; % residuals variance matrix
btrue   = [5 10]';       % parameters of interest

% generating random sample
x1  = exprnd(lambda(1),N,2);    % t=1
x2  = exprnd(lambda(2),N,2);    % t=2
v   = mvnrnd([0; 0],Sigma,N);

% observable sample
X   = reshape([x1'; x2'],2,N*2)';
y   = X*btrue+vec(v');

% POLS
[t_p,St_p,V_p,bhat_p,S_p,D_p]=residual_boots(y,X,N,'n_replic',B);
Sbeta_p   = (X'*X)\((X'*kron(eye(N),Sigma)*X)/(X'*X));
aux       = bsxfun(@minus,t_p,mean(t_p));
Sbeta_bp  = aux'*aux/B;

% GLS: Random effect (Sigma is known)
Pond    = Sigma^(-1/2);
Xtilde  = kron(eye(N),Pond)*X;
ytilde  = kron(eye(N),Pond)*y;
[t_re,St_re,V_re,bhat_re,S_re,D_re]=residual_boots(ytilde,Xtilde,N,'n_replic
    ',B);
Sbeta_re  = inv(Xtilde'*Xtilde);
aux       = bsxfun(@minus,t_re,mean(t_re));
Sbeta_bre = aux'*aux/B;

% Computing Kernels
[fp,Fp,bgrid_p]=mykernel(t_p,'kernel',2);
[fre,Fre,bgrid_re]=mykernel(t_re,'kernel',2);

[X1p,X2p] = meshgrid(bgrid_p(:,1),bgrid_p(:,2));
[X1re,X2re] = meshgrid(bgrid_re(:,1),bgrid_re(:,2));
```

Notice that there is a previous data management of getting the special form of the matrices `X` and `y` before using `residual_boots.m` function. These has stack form:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_N \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \tag{2.8}$$

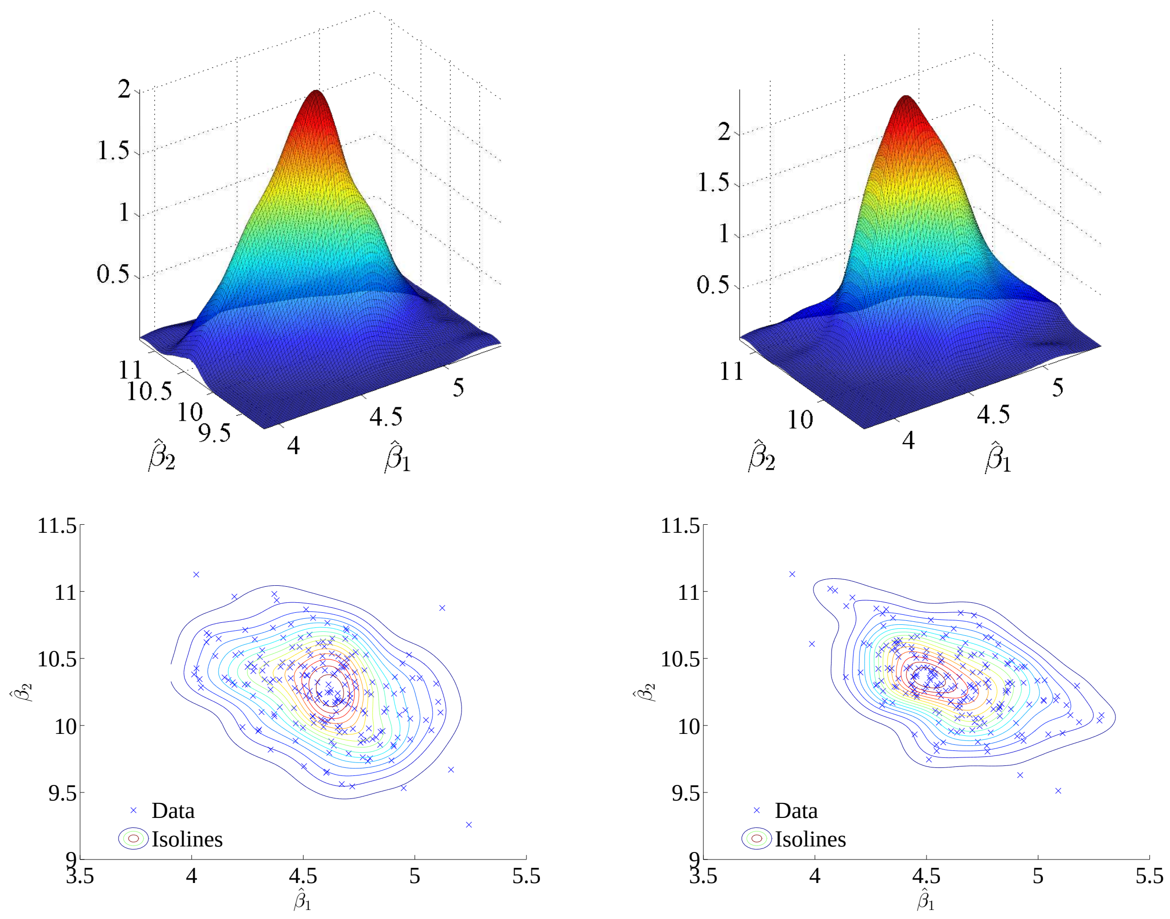
Since, Σ is not diagonal we must expect: a) POLS and GLS (RE) estimators are different, b) GLS (RE) estimators are more precise than POLS, and c) since random sample of $\mathbf{X}_i$ and $\mathbf{v}_i$ are independent from each other, both POLS and GLS (RE) are consistent. Figure 5 shows kernel density estimations based on bootstrap samples for *POLS* and *GLS* estimator. Then, letters a) and b) are confirmed.

TABLE 2. *Standard deviation: comparing asymptotic and bootstrap estimator*

Parameter	**POLS**			**GLS (RE)**		
	POINT EST.	STANDARD DEV. *asymptotic*	STANDARD DEV. *bootstrap*	POINT EST.	STANDARD DEV. *asymptotic*	STANDARD DEV. *bootstrap*
β_1	4.5807	0.2685	0.2499	4.6035	0.2329	0.2492
β_2	10.2943	0.2824	0.3074	10.3077	0.2452	0.2675
$\mathbb{C}(\hat{\beta}_1, \hat{\beta}_2)$		-0.0355	-0.0316		-0.0264	-0.0343

FIGURE 5. *Kernel estimator of POLS and GLS (RE) estimators: Residual bootstrap*

(a) POLS estimator **(b) GLS (RE) estimator**



2.4 PAIRWISE BOOTSTRAP: `pairwise_boots.m`

Another way to obtain bootstrap samples in regression model is getting resamples with replacement directly from the observed data sample. Bootstrapping vectors simply treats the vector that includes the dependent variable and the regression covariates, $(\mathbf{y_i}, \mathbf{X}_i)$, as though they were independent and identically distributed random vectors (possibly with a correlation structure). Then, the bootstrap samples can be used as the data to generate the bootstrap estimates. One important difference to residual bootstrap is that **pairwise bootstrap is less sensitive to the model specification**.

Algorithm 6. Pairwise bootstrap algorithm. *Consider a linear model in the form of (2.7). Assume the we are using an estimator of $\boldsymbol{\beta}$, $T_N(.)$. Do:*

```
1. Using the vector sample $\{\mathbf{y}_i, \mathbf{X}_i\}_{i=1}^N$, estimate the joint EDF $\hat{F}_N(\mathbf{y}, \mathbf{X})$.

   Then, for $b = 1$ to $B$:
2. Obtain a draw from $\hat{F}_N(.)$ and call it $\{[\mathbf{y}_1^{(b)}, \mathbf{X}_1^{(b)}], \ldots, [\mathbf{y}_N^{(b)}, \mathbf{X}_N^{(b)}]\}$,
3. Compute and save, $\hat{\boldsymbol{\beta}}^{(b)} = T_N((\mathbf{y}_1^{(b)}, \mathbf{X}_1), \ldots, (\mathbf{y}_N^{(b)}, \mathbf{X}_N))$,
4. $b = b + 1$.
```

`pairwise_boots.m` function implements algorithm 6.

```
[beta,Yb,Xb]=pairwise_boots(y,X,nobs,varargin)

% Obtains parameter estimator resamples using pairwise bootstrap method
% in a linear model:
%
%    y_i = X_i*beta + v_i
%
% where y_i is a (m x 1) random vector, X_i is (m x k), v_i is (m x 1), and
   beta is the (k x 1) parameter vector.
% ==============
%  Syntax:
% ==============
%       [beta,Yb,Xb]=residual_boots(y,X,nobs)
%       [beta,Yb,Xb]=residual_boots(y,X,nobs,...)
%
% ==============
%   Inputs
% ==============
%  y:   is a stack vector of y_i for i=1 to nobs (size: m.nobs x 1 )
%  X:   is a stack vector of X_i for i=1 to nobs (size: m.nobs x k )
%  nobs: is the number of observations in sample
%   **********
%    Options
%   **********
%   [1] 'n_replic': scalar that sets the number of bootstrap samples. [
   default: B=100]
% ==============
%  Outputs
% ==============
%  beta : (B x k) matrix with vector parameter estimations using bootstrap
   sample.
%  Yb   : (nobs.m x B) object with dependent variable bootstrap sample.
%  Xb   : (nobs.m x k x B) object with covariates bootstrap sample.
%============================================================
```

Example 5. **(Continuation)**. Using the same database that in example 4, I will compare residual and pairwise bootstrap.

```
%% [V] Pairwise bootstrap (Panel data, continuation)
rng(1115);
% POLS
tpair_p = pairwise_boots(y,X,N,'n_replic',B);
aux      = bsxfun(@minus,tpair_p,mean(tpair_p));
Sbetapair_bp = aux'*aux/B;

% GLS: Random effect (Sigma is known)
tpair_re = pairwise_boots(ytilde,Xtilde,N,'n_replic',B);
aux      = bsxfun(@minus,tpair_re,mean(tpair_re));
Sbetapair_bre = aux'*aux/B;

% Computing Kernels
[fp,~,bgrid_p]=mykernel(tpair_p,'kernel',2);
[fre,~,bgrid_re]=mykernel(tpair_re,'kernel',2);

[X1p,X2p] = meshgrid(bgrid_p(:,1),bgrid_p(:,2));
[X1re,X2re] = meshgrid(bgrid_re(:,1),bgrid_re(:,2));
```

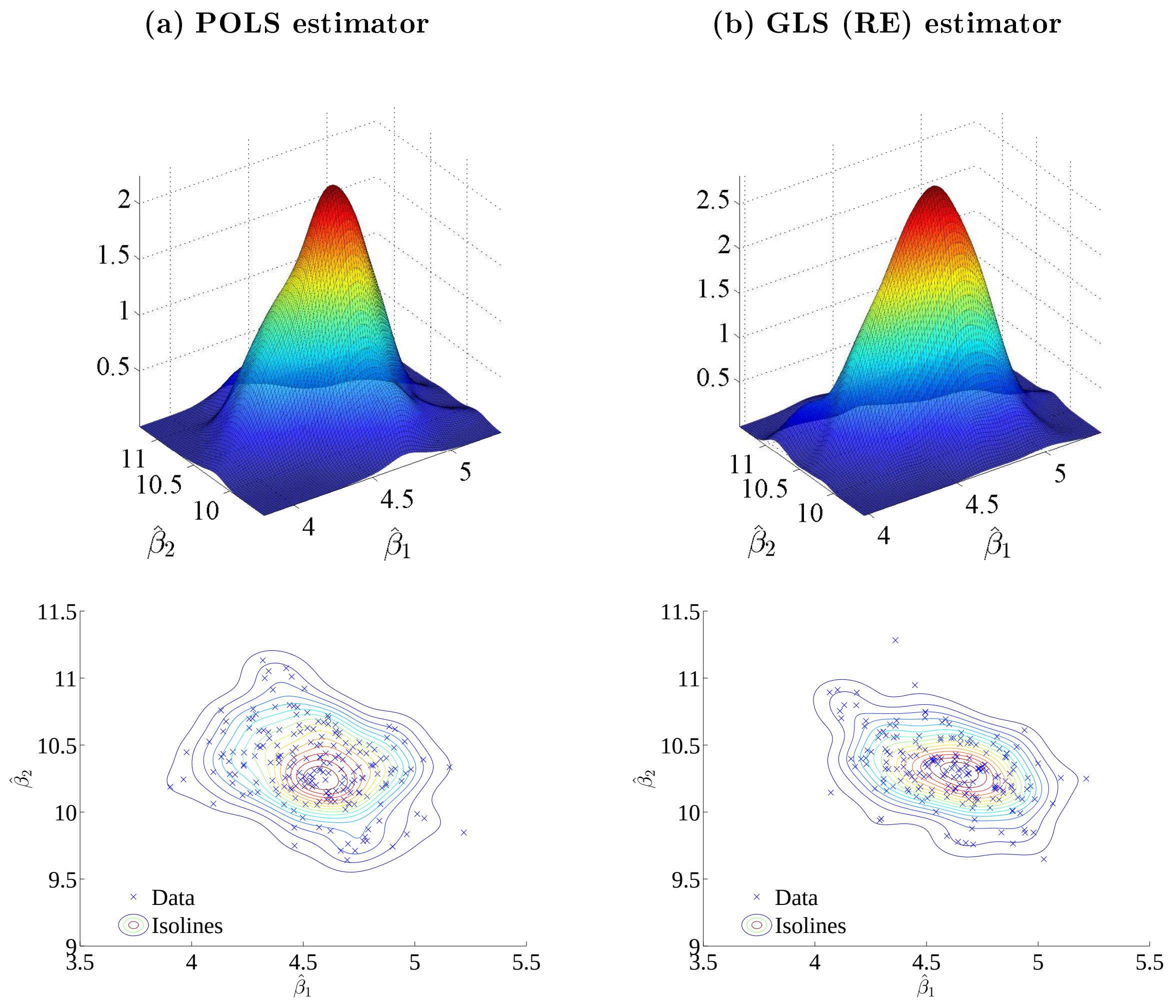
From Tab. 3 we can see that both bootstrap methods give similar estimations and, in the specific case, pairwise bootstrap produce less correlated draws, see all Fig. 6.

TABLE 3. *Standard deviation: comparing residual and pairwise bootstrap*

Parameter	POLS		GLS (RE)	
	residual boots	*pairwise boots*	*residual boots*	*pairwise boots*
β_1	0.2499	0.2360	0.2492	0.2315
β_2	0.3074	0.2774	0.2675	0.2436
$\mathbb{C}(\hat\beta_1, \hat\beta_2)$	-0.0316	-0.0200	-0.0343	-0.0261

Note. boots: bootstrap estimator.

FIGURE 6. *Kernel estimator of POLS and GLS (RE) estimators: Pairwise bootstrap*



2.5 WILD BOOTSTRAP (HETEROSCEDASTICITY): `wild_boots.m`

When residual are heteroscedastic, pairwise bootstrap can do a better job than residual bootstrap. Another approach to deal with heteroscedasticity that bootstrap modified residuals is called the wild bootstrap. There are many variants of this bootstrap and has been shown to be effective in the heteroscedastic case and better than bootstrapping pairs. In general, wild bootstrap follows:

Algorithm 7. Wild bootstrap algorithm. *Consider a linear model in the form of (2.7), however, let $\mathbb{E}[\mathbf{v}_i\mathbf{v}_i'|\mathbf{X}_i] = \Sigma(\mathbf{X}_i)$. Assume the we are using an estimator of $\boldsymbol{\beta}$, $T_N(.)$. Do:*

1. *Compute $\hat{\boldsymbol{\beta}} = T_N((\mathbf{y}_1, \mathbf{X}_1), \ldots, (\mathbf{y}_N, \mathbf{X}_N))$.*

Then, for $b = 1$ to B:

2. *Draw a independent sequence $(u_1^{(b)}, \ldots, u_N^{(b)})$ with zero mean and unity variance (auxiliary variables).*
3. *Construct the bootstrap sample: $\mathbf{y}_i^{(b)} = \mathbf{X}_i\hat{\boldsymbol{\beta}} + u_i^{(b)} f(\mathbf{v}_i)$ for $i = 1, \ldots, N$. Note! Here $f(.)$ is a transformation function.*
4. *Compute and save, $\hat{\boldsymbol{\beta}}^{(b)} = T_N((\mathbf{y}_1^{(b)}, \mathbf{X}_1), \ldots, (\mathbf{y}_N^{(b)}, \mathbf{X}_N))$,*
5. *$b = b + 1$.*

wild_boots.m function implements three variants of this algorithm:

1. **Rademacher distribution F2**: auxiliary variables are 1 with probability $p = 1/2$ and -1 with measure $1 - p$.
2. **Mammen's two-point distribution, F1**: auxiliary variables are $-(\sqrt{5} - 1)/2$ with probability $p = (\sqrt{5} + 1)/(2\sqrt{5})$ and $(\sqrt{5} + 1)/2$ with measure $1 - p$.
3. **Gaussian distribution**. Obtain auxiliary series from a standard normal distribution.

In any case, the option $HC = 0$ implies $f(x) = x$, while $HC = 1$ ask for using:

$$f(\mathbf{v}_i) = (\mathbf{I} - \mathbf{P}_i)^{-1/2}\mathbf{v}_i \tag{2.9}$$

where $\mathbf{P}_i$ is the ith (m, m) diagonal square block of $\mathbf{P} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$.

```
[beta,V,bhat,S,D]=wild_boots(y,X,nobs,varargin)

% Obtains parameter estimator resamples using wild bootstrap method in a
   linear model:
%
%    y_i = X_i*beta + v_i
%
% where y_i is a (m x 1) random vector, X_i is (m x k), v_i is (m x 1),  and
   beta is the (k x 1) parameter vector.
% ==============
%  Syntax:
% ==============
%    [beta,V,bhat,D]=wild_boots(y,X,nobs)
%    [beta,V,bhat,D]=wild_boots(y,X,nobs,...)
%
% ==============
%   Inputs
% ==============
%  y:   is a stack vector of y_i for i=1 to nobs (size: m.nobs x 1 )
%  X:   is a stack vector of X_i for i=1 to nobs (size: m.nobs x k )
%  nobs: is the number of observations in sample
%
%   *********
%    Options
%   *********
%   [1] 'n_replic'  : scalar that sets the number of bootstrap samples. [
   default: B=100].
%   [2] 'size'      : scalar that sets the size of each bootstrap sample. [
   default: N=nobs].
%   [3] 'algorithm' : scalar (1, 2, or 3) that sets the algorithm used to
   compute auxiliar variable. 1, implements  Rademacher distribution. 2,
   implements Mammmen's two  point distribution. 3, uses a Gaussian
   distibution. [default: 1].
```

```
%   [4] 'HC'         : logic (0 or 1) which specifies residual transformation
    function. 0, does not make any transformation. 1, uses f(e)=(1-h)^0.5*e. [
    default: 0]
%
% ==============
%  Outputs
% ==============
%  beta : (B x k) matrix with vector parameter estimations using bootstrap
    sample.
%  V    : (nobs x m x B) object with residual bootstrap sample (transformed).
%  bhat: OLS estimation using the given sample.
%  S    : residual variance estimation using the given sample.
%  D    : residuals estimation using OLS.
%============================================================
```

Example 6. Now assume that $\mathbf{v}_i = diag([x_{i,11}; x_{i,21}])\mathbf{e}_i$, where $\mathbf{e}_i|\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \Sigma)$. Then I am assuming that the errors are heteroscedastic due to a contemporaneous dependence on variable 1. To obtain GLS estimator we must calculate:

$$\begin{aligned}
\mathbb{E}[\mathbf{v}_i\mathbf{v}_i'] &= \mathbb{E}[diag([x_{i,11}; x_{i,21}])\mathbf{e}_i\mathbf{e}_i' diag([x_{i,11}; x_{i,21}])] \\
&= \mathbb{E}[diag([x_{i,11}; x_{i,21}]) \times \Sigma \times diag([x_{i,11}; x_{i,21}])] \\
&= \mathbb{E}\Bigg\{ \begin{bmatrix} \omega_{11}x_{i,11}^2 & \omega_{i,12}x_{i,11}x_{i,21} \\ \omega_{21}x_{i,11}x_{i,21} & \omega_{22}x_{i,21}^2 \end{bmatrix} \Bigg\} \\
&= \begin{bmatrix} 2\omega_{11}\lambda_1^2 & \omega_{12}\lambda_1\lambda_2 \\ \omega_{21}\lambda_1\lambda_2 & 2\omega_{22}\lambda_2^2 \end{bmatrix}
\end{aligned}$$

Hence,

$$\Omega = \mathbb{E}[\mathbf{v}_i\mathbf{v}_i'] = \begin{bmatrix} 2\omega_{11}\lambda_1^2 & \omega_{12}\lambda_1\lambda_2 \\ \omega_{21}\lambda_1\lambda_2 & 2\omega_{22}\lambda_2^2 \end{bmatrix} \qquad (2.10)$$

.

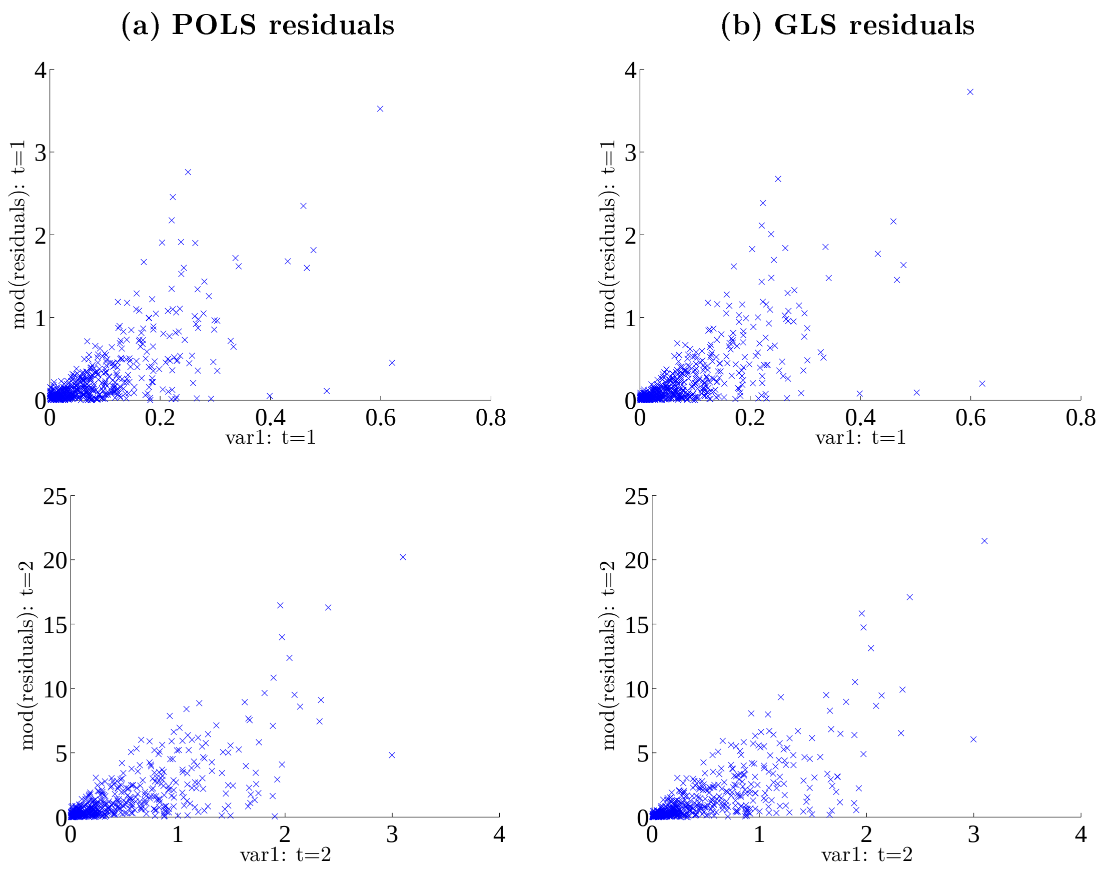
```
%% [VI] Wild bootstrap (heteroscedasticity)
rng(1115);   % for reproducibility
% x1 : variables observed at t=1
% x2 : variables observed at t=2

% observable sample
v     = ([x1(:,1) x2(:,1)]).*mvnrnd([0; 0],Sigma,N);
y   = X*btrue+vec(v');
Omg   = [2 1;1 2].*(diag(lambda)*Sigma*diag(lambda));
Pond    = Omg^(-1/2);
Xtilde  = kron(eye(N),Pond)*X;
ytilde  = kron(eye(N),Pond)*y;

% POLS
[tp_r,~,~,bhat_p,~, Rp]= residual_boots(y, ...
                            X,N,'n_replic',B);
aux    = bsxfun(@minus,tp_r,mean(tp_r));
Sbp_r  = aux'*aux/B;

tp_p   = pairwise_boots(y,X,N,'n_replic',B);
aux    = bsxfun(@minus,tp_p,mean(tp_p));
Sbp_p  = aux'*aux/B;

tp_w  = wild_boots(y,X,N,'n_replic',B,'algo',1);
aux    = bsxfun(@minus,tp_w,mean(tp_w));
Sbp_w  = aux'*aux/B;

% GLS estimator
[tg_r,~,~,bhat_g]  = residual_boots(ytilde, ...
                              Xtilde,N,'n_replic',B);
Rgp   = reshape(y-X*bhat_g,2,N)';
aux   = bsxfun(@minus,tg_r,mean(tg_r));
Sbg_r = aux'*aux/B;

tg_p  = pairwise_boots(ytilde,Xtilde,N,'n_replic',B);
aux   = bsxfun(@minus,tg_p,mean(tg_p));
Sbg_p = aux'*aux/B;

tg_w  = wild_boots(ytilde,Xtilde,N,'n_replic',B,'algo',1);
aux   = bsxfun(@minus,tg_w,mean(tg_w));
Sbg_w = aux'*aux/B;

% asymptotic estimators
Xr    = bsxfun(@times,x1,Rp(:,1)) + bsxfun(@times,x2,Rp(:,2));
Sa_p  = (X'*X)\((Xr'*Xr)/(X'*X));

Xa = permute(reshape(Xtilde',2,2,N),[3 1 2]);
Rgp2 = reshape(kron(eye(N),Pond)*vec(Rgp'),2,N)';

Xgr   = bsxfun(@times,Xa(:,:,1),Rgp2(:,1)) + bsxfun(@times,Xa(:,:,2),Rgp2
    (:,2));
Sa_g  = (Xtilde'*Xtilde)\((Xgr'*Xgr)/(Xtilde'*Xtilde));
```

Figure 7 inspect evidences of heteroscedasticity in both POLS and GLS estimator. Clearly, there is some dependence between x_{t1} and $\hat{v}_t$ for $t = 1, 2$. Hence, instead of using residual bootstrap we can use pairwise or wild bootstrap. Table 4 shows that residual bootstrap tends to underestimate the standards error for $\hat{\beta}_1$ while for $\hat{\beta}_2$ the three bootstrap methods give consistent estimators. The latter results is because of both regressor vectors are independent and $\mathbf{x}_{i2}$ does not influence $\mathbf{v_i}$.

Table 4. *Standard deviation: comparing asymptotic and bootstrap estimators.*

POLS estimator					
Parameter	Point estimator	*Asymp. sd.*	*Residual sd.*	*Pairwise sd.*	*Wild sd.*
$\hat{\beta}_1$	3.9975	0.4777	0.1918	0.4510	0.4452
$\hat{\beta}_2$	10.5535	0.2489	0.1951	0.2544	0.2238
$\mathbb{C}(\hat{\beta}_1, \hat{\beta}_2)$	-	-0.0839	-0.0164	-0.0869	-0.0639
GLS estimator					
Parameter	Point estimator	*Asymp. sd.*	*Residual sd.*	*Pairwise sd.*	*Wild sd.*
$\hat{\beta}_1$	4.4192	0.3605	0.1592	0.3576	0.3780
$\hat{\beta}_2$	10.3216	0.1635	0.1510	0.1504	0.1587
$\mathbb{C}(\hat{\beta}_1, \hat{\beta}_2)$	-	-0.0358	-0.0121	-0.0292	-0.0369

Figure 7. *Rsidual modulus vs regressor*



3 Dependent observations

If observations we want to use for bootstrapping do not satisfy the i.i.d. assumption, then using any of the methods presented before will be incorrect. In this section I will show two ways we can deal with dependent data. The first one is similar to residual bootstrap but adjusted to time series. The second is the well-known moving-block bootstrap. Based on Chernick (1999), Chernick and LaBudde (2011), and Politis and Romano (1994).

3.1 Model-based approach: `VAR_boots.m`

Firs of all, consider a stationary process $\{\mathbf{z}_t\}_{t=1}^{T}$ $(m \times 1)$ approximated by a finite VAR(p) model:

$$\mathbf{z}_t = \mathbf{C_0} + \mathbf{C_1}\mathbf{z}_{t-1} + \cdots + \mathbf{C_p}\mathbf{z}_{t-p} + \mathbf{v}_t \tag{3.1}$$

where $\{\mathbf{v}_t\}_{t=1}^{T}$ is a $(m \times 1)$ serially uncorrelated random vector with $\mathbb{E}[\mathbf{v}_t\mathbf{v}_t'] = \Sigma_v$. Estimations are based on ordinary least square (OLS): define $\mathbf{C} \equiv \begin{bmatrix}\mathbf{C_0} & \mathbf{C_1} & \dots & \mathbf{C_p}\end{bmatrix}$, $\boldsymbol{\beta} = vec(\mathbf{C})$, $\tilde{\mathbf{X}}_t \equiv \begin{bmatrix}1 & \mathbf{z}_t' & \mathbf{z}_{t-1}' & \dots & \mathbf{z}_{t-p+1}'\end{bmatrix}'$, and $\mathbf{X}_t = \tilde{\mathbf{X}}_{t-1}' \otimes \mathbf{I}_m$. Then,

$$\mathbf{z}_t = \mathbf{C}\tilde{\mathbf{X}}_{t-1} + \mathbf{v}_t \tag{3.2}$$

for $t = p+1, \dots, T$. Or equivalently as (2.7) form:

$$\mathbf{z}_t = (\tilde{\mathbf{X}}_{t-1}' \otimes \mathbf{I}_m)vec(\mathbf{C}) + \mathbf{v}_t \tag{3.3}$$

or

$$\mathbf{z}_t = \mathbf{X}_t\boldsymbol{\beta} + \mathbf{v}_t \tag{3.4}$$

for $t = p+1, \dots, T$.

Algorithm 8. Autoregressive bootstrap algorithm. *Consider a linear model in the form of (3.1). Assume the we are using an estimator of $\boldsymbol{\beta}$, $T_N(.)$. Do:*

1. *Compute $\hat{\boldsymbol{\beta}} = T_N((\mathbf{z}_{p+1}, \mathbf{X}_{p+1}), \ldots, (\mathbf{z}_T, \mathbf{X}_T))$, and residuals $\hat{\mathbf{v}}_t = \mathbf{z}_t - \mathbf{X}_t\hat{\boldsymbol{\beta}}$ for $t = p+1, \ldots, T$.*

   Then, for $b = 1$ to B:

2. *Get a sample with replacement from $\{\hat{\mathbf{v}}_t\}_{t=p+1}^T$ and call it $\{\mathbf{v}_t^{(b)}\}_{t=p+1}^T$.*
3. *Construct the bootstrap sample: $\mathbf{z}_t^{(b)} = \mathbf{X}_t^{(b)}\hat{\boldsymbol{\beta}} + \mathbf{v}_t^{(b)}$ for $t = 1, \ldots, T + burn$ with $\mathbf{X}_1^{(b)} = \mathbf{X}_1$. The scalar burn specifies the first observations to not consider in the computation of the statistic.*
4. *Compute and save, $\hat{\boldsymbol{\beta}}^{(b)} = T_N((\mathbf{z}_{p+1}^{(b)}, \mathbf{X}_{p+1}^{(b)}), \ldots, (\mathbf{z}_T^{(b)}, \mathbf{X}_T^{(b)}))$,*
5. *$b = b + 1$.*

Hence, the algorithm is very similar to *residual bootstrap*, however there is an important difference: **covariates must be updated in constructing the artificial sample**. VAR_boots.m implements the autoregressive bootstrap algorithm.

```
[beta,Omg,Z,X,bhat,S,Vsample]=VAR_boots(D,varargin)

% Obtains parameter estimator resamples using parametric bootstrap for VAR
% models.
%
%    z_t = C_0 + C_1*z_{t-1} + ... + C_p*z_{t-p} + v_t
%
% where z_t is a (m x 1) random vector, v_t is (m x 1), and C_i is the (m x m
  ) vector parameter for i=1, ..., p.
% ==============
%  Syntax:
% ==============
%       [beta,Omg,Z,X,bhat,S,Vsample]=VAR_boots(D)
%       [beta,Omg,Z,X,bhat,S,Vsample]=VAR_boots(D,...)
%
% ==============
%   Inputs
% ==============
%  D : Observed sammple of endogenous variables.
%
%   **********
%    Options
%   **********
%   [1] 'n_replic'   : scalar that sets the number of bootstrap samples. [
  default: B=100]
%   [2] 'order'      : scalar that sets the number of lag to be considered. [
  default: p=1]
%   [3] 'noconstant' : indicates to not include constant in estimation. [
  default: const=1]
%   [4] 'wild'       : indicates to use wild bootstrap.
%                      [default: uses residual bootstrap]
%   [5] 'burnin'     : sets how many observations will not be considered in
  parameter estimation. [default: burn=0]
% ==============
%  Outputs
% ==============
%  beta: (B x k) matrix with vector parameter estimations using bootstrap
  sample.
%  Omg : (m x m x B) object with estimation of residual variance matrix.
%  Z    : Stack data for z_t
%  X    : Stack data for z_{t-1},...,z_{t-p}.
%  bhat: OLS estimation using the given sample.
%  S    : residual variance estimation using the given sample.
%  Vsample  : residuals estimation using OLS.
%============================================================
```

Example 7. **Estimating confidence intervals for IRF's in a SVAR model**. Lets study the interdependence of monetary policy and stock markets applying (Bjornland and Leitemo) methodology to Brazilian data. Let $\mathbf{z}_t = [y_t, \pi_t, \Delta c_t, \Delta s_t, i_t]'$ where y_t is the output gap (HP filter for Industrial Production Index), π_t denotes annual inflation (twelve-months change in CPI), Δc_t commodities inflation (twelve-months change in $\mathsf{IC-Br}$), Δs_t real stock price variation (monthly change in $\mathsf{IBOVESPA}$) and i_t denotes interest rate (SELIC). Then specifying a reduced form model as in (3.1) with $p = 2$, the $\mathsf{MA}(\infty)$ representation can be written as:

$$\mathbf{z}_t = \mathbf{\Theta}(L)\mathbf{v}_t, \qquad \mathbf{v}_t \sim iid(0, \Sigma_v) \tag{3.5}$$

$$\mathbf{z}_t = \mathbf{\Psi}(L)\boldsymbol{\varepsilon}_t, \qquad \boldsymbol{\varepsilon}_t \sim iid(0, \mathbf{I}_m) \tag{3.6}$$

$$\mathbf{v}_t = \mathbf{A}\boldsymbol{\varepsilon}_t \tag{3.7}$$

$$\mathbf{\Psi}(L) = \mathbf{\Theta}(L)\mathbf{A} \tag{3.8}$$

It is imposed both short-run and long-run restriction for identification of $\mathbf{A}$:

1. **Short run restrictions**. Recursivity of monetary policy and real stock price shows over the rest of the variables is assumed, i.e.,

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \tag{3.9}$$

   The main difference from related studies is that BL does not assume $a_{45} = 0$ permitting simultaneity between interest rate and real stock price shocks. In this way, BL impose nine zero restrictions.

2. **Long-run restrictions**. Based on conventional long-run neutrality for monetary policy, it is imposed that interest rate shocks have no long-run effects on the level of real stock prices ($\mathsf{s}_t = \ln \frac{\mathsf{S}_t}{\mathsf{P}_t}$). The cumulated long-run effect of ε_t^{mp} on Δs_t (or, equivalently, the long-run effect on s_t) is quantified by the $(4, 5)$ element of the matrix $\mathbf{\Psi}(1)$, i.e.,

$$\mathbf{\Psi}_{45}(1) = \mathbf{\Theta}_{44}(1)a_{45} + \mathbf{\Theta}_{45}(1)a_{55} \tag{3.10}$$

   Hence, the last restriction is a linear dependency between a_{45} and a_{55}:

$$a_{55} = -\frac{\mathbf{\Theta}_{44}}{\mathbf{\Theta}_{45}}a_{45} \tag{3.11}$$

So, these restrictions can be specified as:

$$vec(\mathbf{A}) = \mathbf{R}_A\boldsymbol{\gamma}_A \tag{3.12}$$

where $\boldsymbol{\gamma}_A$ is the parameter vector with free variation.

```
%% [VII] VAR Bootstrap
rng(1115);             % seed for reproducibility
load dataVAR.mat;      % DD contains data used in VAR
p = 2;                 % lag order
B = 500;               % bootstrap replications
```

```matlab
K = 50;              % impulse responses horizon

% Bootstrapping: wild bootstrap
[bhat,Omg,Z,X,b_est,S]=VAR_boots(DD,'order',p,'n_replic',B,'burnin',20,'wild
    ');
[T,m]=size(DD);

% Initialization (media) [Long-run restriction]
BET       = reshape(b_est,m,[]);
LR        = inv(eye(m)-BET(:,2:m+1)-BET(:,m+2:2*m+1));
RA(m^2,m*(m+1)/2) = -LR(4,4)/LR(4,5);
gamma0 = 0.8*ones(m*(m+1)/2,1);
gamma0(m*(m+1)/2) = -gamma0(m*(m+1)/2);
gamma0 = mlSVAR(RA,gamma0,S,T,'tau',0.3, ...
             'errotol_score',1e-7,'errortol_grad',1e-7);
F   = [BET(:,2:end); ...
        [kron(eye(p-1,p-1),eye(m,m)), zeros(m*(p-1),m)]];
A   = reshape(RA*gamma0,m,m);
IR  = VARimpulse(F,A,K);

% Bootstrap for IRF's [SVAR]
IRb = [];
for b=1:B
    beta   = bhat(:,b);
    BET    = reshape(beta,m,[]);
    Sb     = Omg(:,:,b);
    F = [BET(:,2:end);...
        [kron(eye(p-1,p-1),eye(m,m)), zeros(m*(p-1),m)]];
    eigen = eig(F);           % is stable?
    if ~any(abs(eigen)>1)
        LR     =  inv(eye(m)-BET(:,2:m+1)-BET(:,m+2:2*m+1));
        % Long-run restriction
        RA(m^2,m*(m+1)/2)  = -LR(4,4)/LR(4,5);
        gammaAb  = mlSVAR(RA,gamma0,Sb,T,'tau',.3, ...
                'errotol_score',1e-4,'errortol_grad',1e-6,'quiet');
        Ab      = reshape(RA*gammaAb,m,m);
        IRb(:,:,:,b)  = VARimpulse(F,Ab,K);
    end
end
```

The function `VAR_boots.m` does not compute other statistic than the OLS estimations for $\boldsymbol{\beta}$. The reason for this is that, in time series analysis, we are commonly interested on impulses response functions, variance decomposition, or historical decomposition of structural shocks, however any of these statistic needs a previous structural estimation for $\mathbf{A}$ which is obtained using different identification strategies. In the example under analysis, structural VAR estimation is based on quasi maximum likelihood subject to identification restriction listed above. This section could last some minutes than the others due to scoring algorithm for computing $\boldsymbol{\gamma}_A$. Confidence intervals for impulse response functions for both monetary policy shocks and stock price shocks are depicted in Fig. 8.

3.2 Stationary Moving-Block Bootstrap: `MB_boots.m`

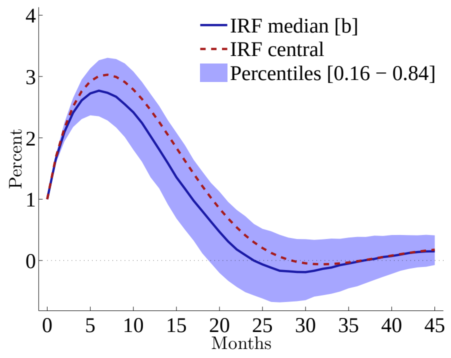
Another approach to obtain appropriate bootstrap samples from dependent data is using a non-parametric one. The basic idea is to divide the entire sample in blocks of observations of size R, bootstrap regarding each of these blocks as one observation, and finally, compute the statistic of interest with the artificial sample.

Algorithm 9. Stationary moving-block bootstrap. *Given a sample $\mathbf{x}_1, \ldots, \mathbf{x}_T$ with possible serial correlation and an statistic of interest $S_T(.)$, then for $b = 1$ to B:*

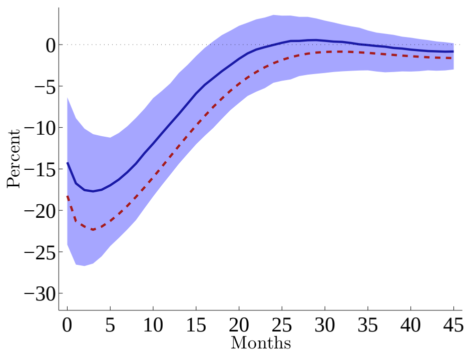
Figure 8. *Monetary policy and stock price shocks - SVAR*

Monetary policy shock

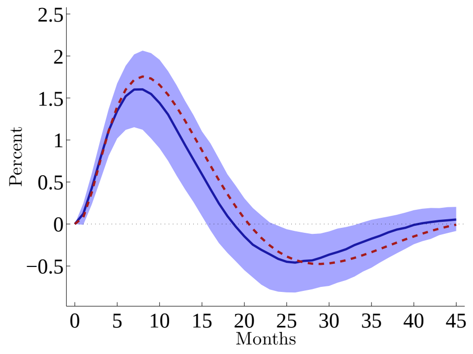
(a) Interest rate



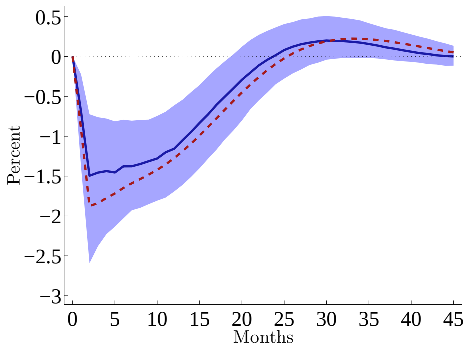
(b) Real stock prices



(c) Annual inflation

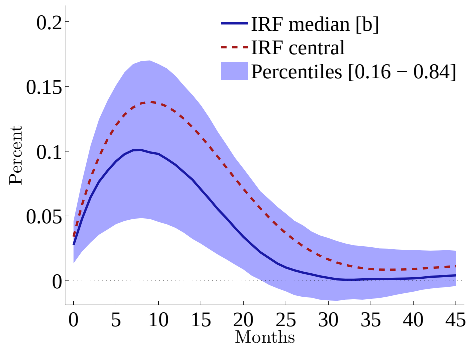


(d) Output (HP cycle)

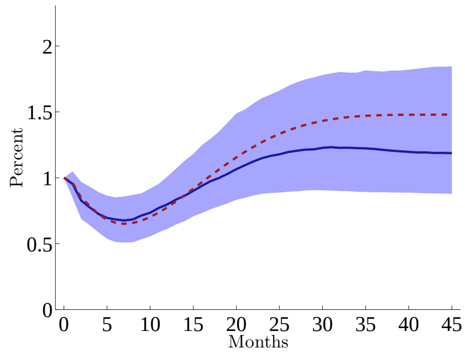


Stock price shock

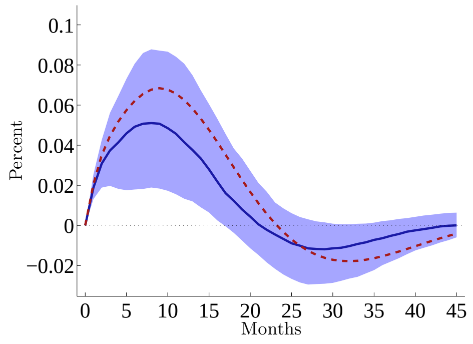
(e) Interest rate



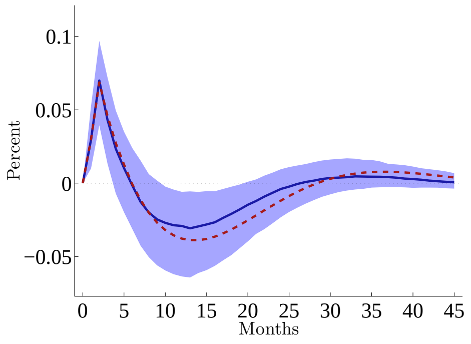
(f) Real stock prices



(g) Annual inflation



(h) Output (HP cycle)



Note. 500 replications were used. Central IRF is obtained using observed sample, while median IRF is obtained computing median in the IRF's in the bootstrap sample. Shocks are normalized so that monetary policy shock increases interest rate by one percent immediately, while stock price shock increases the real stock prices with one percent instantaneously.

1. *Draw an i.i.d. sequence from a geometric distribution with probability p:* $l_1, l_2, \dots$.
2. *Draw an i.i.d. sequence from discrete uniform distribution on* $1, \dots, T$*:* $t_1, t_2, \dots$.
3. *Construct the sequence of blocks,* $B_{t,n}$*:*

$$B_{t,l} = \{\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+l-1}\} \tag{3.13}$$

4. *Construct the pseudo time series:* $\mathbf{x}_1^{(b)}, \dots, \mathbf{x}_T^{(b)}$*, by sorting blocks* $\{B_{t_1,l_1}, B_{t_2,l_2}, \dots\}$ *until obtain* T *columns.*
5. *Compute the statistic of interest,* $S_T^{(b)} = S_T(\mathbf{x}_1^{(b)}, \dots, \mathbf{x}_T^{(b)})$.
6. $b = b + 1$.

In order to implement stationary moving block bootstrap in the present package:

```
[statboot,Dboot]=MB_boots(D,lambda,stat,varargin)

% Executes moving-block bootstrap using D [tobs x m matrix].
% ==============
%  Syntax:
% ==============
%       [statboot,Dboot]=MB_boots(D,lambda,stat)
%       [statboot,Dboot]=MB_boots(D,lambda,stat,...)
%
% ==============
%   Inputs
% ==============
%  D       : Observed sample.
%  lambda  : mean of the size for each block.
%  stat    : function handle with statistic formula.
%
%   **********
%     Options
%   **********
%   [1] 'n_replic'   : scalar that sets the number of bootstrap samples.
   [default: B=100]
%   [2] 'size'       : scalar that sets the size of each resample. [
   default: T=tobs].
% ==============
%  Outputs
% ==============
%  statboot: (k x B) matrix with bootstrap sample of the statistic of
   interest.
%  Dboot   : (T x m x B) object with resamples.
%==========================================================
```

Example 8. **Central limit theorem for dependent process.** The objective of this exercise is to show the inconsistency of standard error estimator based on naive bootstrap when we are dealing with dependent data and, at the same time, how moving-block bootstrap corrects for dependency. Let $\{y\}$ be a stochastic process such that:

$$y_t = \mu + \sum_{i=0}^{\infty} \psi_i \varepsilon_{t-i}$$
$$y_t = \mu + \Psi_\infty(L)\varepsilon_t$$

where $\psi_0 = 1$. Then, under some regularity conditions:

$$\sqrt{T}(\overline{y}_T - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma_\varepsilon^2 \Psi_\infty(1)^2) \tag{3.14}$$

where $\overline{y}_T = T^{-1}\sum_{t=1}^T y_t$ and ε_t is an i.i.d process with variance equal to σ_ε^2. This theorem predicts a different asymptotic variance for $\sqrt{T}\overline{y}_T$ than in the traditional Lindeberg-Lévy central limit theorem. For example, let

$$y_t = \mu(1-\rho) + \rho y_{t-1} + \varepsilon_t \tag{3.15}$$

with $|\rho| < 1$, then $\Psi_\infty(L) = (1-\rho L)^{-1}$. The asymptotic standard deviation for $\sqrt{T}(\overline{y}_T - \mu)$ is $\frac{\sigma_\varepsilon}{1-\rho}$. Then using, for example, *naive bootstrap* when data seems to be serially correlated is incorrect.

```
%% [VIII] Moving-Block bootstrap
rng(1115);      % for reproducibility
B     = 500;
T     = 200;
p     = 1;
mu    = 2;
sigma = 2;
rho   = 0.4;
y     = nan(T,1);
y(1)  = 0;
for t=2:T
    y(t)=mu*(1-rho)+rho*y(t-1)+ sigma*randn;
end

lambda = T^1/3;

% MB bootstrap and naive bootstrap
statMB  = MB_boots(y,lambda,@mean,'n_replic',B)';
statNai = naive_boots(y,@mean,'n_replic',B);

% Printing results
rr=[sigma/(1-rho) std([sqrt(T)*(statNai-mean(y)) sqrt(T)*(statMB-mean(y))])
    ];
fprintf('\n=================================\n');
fprintf('True  Naive boots  M-B boots\n');
fprintf('=================================\n');
fprintf('%2.4f   %2.4f    %2.4f \n',rr);
fprintf('=================================\n');
```

The results of this exercise are:

```
==================================
True  Naive boots  M-B boots
==================================
3.3333   2.0561    3.0301
==================================
```

So, there is evident that moving-block corrects the standard error estimator.

4 List of functions

The present document shows how to apply different bootstrap methods to different data structure using MATLAB functions scripts from my bootstrap package.

1. naive_boots.m.
2. nonparam_boots.m.
3. residual_boots.m.
4. pairwise_boots.m.
5. wild_boots.m.
6. VAR_boots.m.
7. MB_boots.m.

   Along the document I have used other functions which do not belong to my bootstrap package.

8. mykernel.m.
9. mlSVAR.m.
10. VARimpulse.m

References

Cameron, A. and Miller, D. (2015). A practitioner's guide to cluster-robust inference. *Journal of Human Resources*, 50(2):317–372.

Cameron, A. and Trivedi, P. (2005). *Microeconometrics: Methods and Applications*. Cambridge University Press.

Chernick, M. (1999). *Bootstrap Methods: A Practitioner's Guide*. Wiley Series in Probability and Statistics. Wiley.

Chernick, M. and LaBudde, R. A. (2011). *An Introduction to Bootstrap Methods with Applications to R*. Wiley.

Horowitz, J. L. (2001). The bootstrap. In Heckman, J. and Leamer, E., editors, *Handbook of Econometrics*, volume 5, pages 3159–3228. Elsevier Science.

Politis, D. and Romano, J. (1994). The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313.

Appendix

Kernel estimators

Given a kernel function $k(.)$, the kernel density estimator of $\mathbf{x}$ with q continuous variables, $\mathbf{x}^c$, and r discrete variables, $\mathbf{x}^d$, is

$$\hat{f}_N(\mathbf{x}^c,\mathbf{x}^d) = (Nh_1\dots h_q)^{-1}\sum_{i=1}^{N}\prod_{j=1}^{q}k[(X^c_{i,j}-x^c_j)/h_j]\prod_{s=1}^{r}l(\mathbf{x}^d_i,u) \tag{.1}$$

For bootstrapping we are more interested in estimation of the cumulative distribution function rather than in the density function, then we must integrated the kernel density estimator over the support of the random vector. Before integrating it will be useful to compute

$$\begin{aligned}\int_{-\infty}^{x}k\Big(\frac{x_i-v}{h}\Big)dv &= -h\int_{\infty}^{(x_i-x)/h}k(\psi)d\psi\\ &= h\int_{(x_i-x)/h}^{\infty}k(\psi)d\psi\\ &= h\int_{-\infty}^{-(x_i-x)/h}k(\psi)d\psi\\ &= hG\Big(-\frac{x_i-x}{h}\Big)\end{aligned}$$

hence,

$$\int_{-\infty}^{x^c_j}k\Big(\frac{x_i-v}{h_j}\Big)dv = h_jG\Big(\frac{x-x_i}{h_j}\Big) \tag{.2}$$

where $G(\psi)=\int_{-\infty}^{\psi}k(v)dv$. Now, given the structure of the random vector, the estimator of interes is:

$$\hat{F}_N(\mathbf{x}^c,\mathbf{x}^d) = \sum_{u\le\mathbf{x}^d}\int_{-\infty}^{\mathbf{x}^c}\hat{f}_N(\mathbf{x}^c,\mathbf{x}^d) \tag{.3}$$

Substituting (.2) into this expression we obtain:

$$\hat{F}_N(\mathbf{x}^c,\mathbf{x}^d) = N^{-1}\sum_{i=1}^{N}\Big(\prod_{j=1}^{q}G[(x^c_j-X^c_{i,j})/h_j]\Big)\Big(\sum_{u\le\mathbf{x}^d}L(\mathbf{x}^d_i,u)\Big) \tag{.4}$$