Borland Pascal Tools for Bootstrapping Dependent Data

A. MOUIHA
Dépt. de Mathématiques.
Faculté des Sciences Dhar Mahraz.
Fès, Maroc.

Abstract

One of the drawbacks of the bootstrap method is that it is time consuming when exucuting its applications, especielly with some statistical software (example: S-plus, Minitab, Gauss..). But when using the programming languages (example: Borland Pascal, C/C++,..) this execution becomes faster. In this paper, we describe some Borland Pascal procedures, functions and units to overcome this problem.

Key Words: Borland Pascal, S-Plus, Blocks bootstrap, Bootstrap for stationary processes.

1 Introduction

The bootstrap is a powerful tool for inference in situation where standard approximations are not available. It was introduced in the context of non-parametric analysis of *iid* samples (Efron, 1979), but much research into its use in more complicated settings has followed. Künsch (1989), Mouiha and Rais (2000) provide an excellent reference on the uses of bootstrap and related methods. In conjunction with that, I have written a Borland Pascal program to implement these methods. Davison and Hinkley (1997) have written other bootstrap programs using S-Plus.

Suppose that we have a sample $\chi = (X_1, ..., X_n)$ come from some n-dimensional distribution F and interest is in a functional, $\theta = t(F_n)$, of that distribution, where F_n is the empirical distribution of χ. Inference for θ is often based on assumptions such that F_n has some specific form which depends on a finite number of unknown parameters. If the assumed form of F_n and the functional $t(.)$ are sufficiently simple, then exact or asymptotic distribution theory can be used for inference.

The bootstrap and other resampling methods are designed to allow inference

*Correspondence to: B.P: 99, Taounate 34000, Maroc.

to proceed with fewer assumptions on the distribution or the functional of interest. They are also useful when the simple size n is too small. See Mouiha and Rais (2000).
Suppose that we are in nonparametric case and the data are *iid*, then $F(x_1, ..., x_n) = \prod_{i=1}^{n} F(x_i)$ and F is estimated by the empirical distribution function $\hat{F}$. θ is then estimated by $\hat{\theta} = t(\hat{F}_n)$. Suppose now that χ^* is a dataset with distribution $\hat{F}_n$ and $\hat{\theta}^*$ is the corresponding estimate of $\hat{\theta}$. The bootstrap uses $\hat{\theta}^* - \hat{\theta}$ for inference on $\hat{\theta} - \theta$. Then it is usually necessary to generate many replicates of $\hat{\theta}^* - \hat{\theta}$ and then use Monte Carlo methods for inference.

2 Bootstrap for dependent observations

The ordinary bootstrap will not work for time series as the observations are clearly not *iid*. It is necessary to make some allowance for the autocorrelation between observations and one way for doing this is the **block bootstrap (MBB)** (Künsch, 1989) which does not resample individual observations but resamples blocks of length l and fits them together to produce a replicate of length n. Where n is the length of the original dataset and $l = o(n)$. This method , however, does not reproduce the same dependence structure as in the original data because the blocks are not independent. A modification of this last method is **bootstrap for stationary processes (BSP)** (Mouiha and Rais, 2000) which reproduces the essence of this structure at least at dependence order p. That is, we approximate any stationary process by a p^{th} order Markov Chain, estimate the transition densities and generate our *BSP* sample from the transitions density of this Markov chain. In fact, and by decomposition theorem (Brockwell and Davis, 1991), we estimate the order p basing on the original data, saying, $\chi = (X_1, ..., X_n)$ and apply the *BSP* method.

3 An example of Borland Pascal program used in bootstrap

To compare the two last methods, we will describe, in followings, some Borland Pascal procedures, functions and units which will be the basis of the main unit **mbb-bsp**.

- (**unit var-p**) contains all variables used in **mbb-bsp**.
- (**unit stat-p**) describes the statistics used in **mbb-bsp**.

- (**unit calcul-p**) of the influence functions used in the *BSP* method.
- (**unit dens-p**) is required for some densities used in the methods.
- (**unit arma-p**) describes the models used in the simulations.
- Other functions for generating some random numbers are in (**unit alea-p**).
- To generate the blocks *MBB*, we need the (**procedure bloc**).
- To generate the first block used in *MBB*, we need the (**procedure bloc1**).
- Or to generate it by using the normal density, we need the (**procedure rmulti**).
- Generate the *BSP sample* by the (**procedure rcond**).

Program 3.1. (Unit mbb-bsp for *BSP* and *MBB* methods.),

```
{$N+,E+}
unit mbb-bsp;
 interface
 uses  crt, var-p,  stat-p,  calcul-p,  dens-p,  arma-p, alea-p;
     procedure bloc(l,n:integer;x:vecteur;var x1:vecteur);
     procedure bloc1(l,n:integer;x:vecteur;var x1:vecteur);
     procedure rmulti(l,n:integer;h1:extended;x:vecteur;var x2:vecteur);
     procedure rcond(l,n:integer;h1:extended;a:vecteur; var x2:vecteur);
implementation
(*************************************************************)
procedure bloc(l,n:integer;x:vecteur;var x1:vecteur); (*Return
resamples from MBB method.
 (*************************************************************)
var j,t,u,b:integer;
begin
      b:=trunc(n/l);
      for j:=1 to b do
      begin
           u:=random((n-l+1));
           for t:=1 to l do
            x1^[(j-1)*l+t]:=x^[u+t];
      end;
end;
```

```
(*****************************************************************)
procedure bloc1;{return one l_block }
(*****************************************************************)
var u,t:integer;
begin
     u:=random(n-l+1);
     for t:=1 to l do
       x1^[t]:=x^[u+l];
end;
(*****************************************************************)
procedure rmulti; (*Generate the first bloc (x(1),..,x(l+1)) using
Normal density. The bloc is X2 and l is the block length.
X:Original Sample*)
 (*****************************************************************)
var   k  :vecteur0;i,p,j   :integer; s,u,s1  :extended;
begin
      if l=0 then              x2^[1]:=resnor(x,h1,n)
      else
      begin
          x2^[1]:=resnor(x,h1,n);
          for p:=1 to l do
          begin
           new(k);
             s:=0;k^[0]:=0;
             for i:=1 to n-p do
             begin
                s1:=1;
                for j:=1 to p do
                     s1:=s1*dnor(0,1,((x2^[j]-x^[i+j-1])/h1));
                     k^[i]:=k^[i-1]+s1;
                     s:=s+s1;
             end;
             for i:=0 to n-p do
               k^[i]:=k^[i]/s;
              u:=random;
              for j:=1 to n-p do
              begin
```

```
                if (u>k^[j-1]) and (u<=k^[j]) then
                    x2^[p+1]:=rnorm(x^[j+p],h1);
              end;
            dispose(k);
          end;
      end;
end;
(************************************************************)
procedure rcond; (*Return an n_sample from an l^{th} order markov
Chain. This is  X2=(x2(1),...,x2(n)). X:original data and X2 the
BSP bootstrap sample*)
 (************************************************************)
var t,i,j :integer; s,u,s1:extended; k :vecteur0;
begin
     if l=0 then         rmulti(l-1,n,h1,a,x2)
     else                bloc1(l,n,a,x2);
   begin
     for t:=0 to n+50-l-1 do
     begin
        new(k);
        s:=0; k^[0]:=0;
        for i:=1 to n-l do
        begin
          s1:=1;
          for j:=1 to l do
            s1:=s1*dnor(0,1,(x2^[j+t]-a^[i+j-1])/h1);
            k^[i]:=k^[i-1]+s1;
            s:=s+s1;
        end;
        for i:=0 to n-l do
            k^[i]:=k^[i]/s;
         u:=random;
         for j:=1 to n-l do
         begin
            if (u>k^[j-1]) and (u<=k^[j]) then
                   x2^[t+l+1]:=rnorm(a^[j+l],h1);
         end;
         dispose(k);
     end;
  end;
```

```
end;
```

3.1 Program's results and discussions

To obtain an initial sample for comparing *MBB* and *BSP*, we simulate the dataset from the two following models:

M1. $AR1$: $X_t = 0.8X_{t-1} + \varepsilon_t$, with ε_t is *iid* random.

M2. TAR: $X_t = 0.528Z_t$,with
Z_t=$(0.9Z_{t-1} + \varepsilon_t)1_{\{Z_{t-1}\leq -2.5\}}$ +$(0.8Z_{t-1} + \varepsilon_t)1_{\{Z_{t-1}> -2.5\}}$,
and ε_t is *iid* random.

We generate two samples $n = 27$ and $n = 512$ according to M1 and M2. We use 1000 bootstrap samples and 5000 Monte carlo's iterations. The results are illustred in *table 3.1.*

Parameters			Method BSP			Method MBB		
Size	Model	True σ_n	E'	SD'	RM'	E'	SD'	RM'
n=27	M1	2.91	2.22	0.93	0.16	1.76	0.62	0.20
n=27	M2	3.17	2.12	0.93	0.20	1.72	0.65	0.25
n=512	M1	3.06	2.80	0.36	0.05	2.76	0.58	0.07
n=512	M2	3.21	2.80	0.61	0.05	2.78	0.60	0.05

$E' = E(\sigma_n^*),\ SD' = SD(\sigma_n^*),\ \ RM' = RMSE(\sigma_n^*)$

Table 3.1: Comparison of BSP versus MBB using **mbb-msp** Borland Pascal program for: $\sigma_n = [var(n^{1/2}median(X))]^{1/2}$.

There is clearly a difference between the two methods because the $RMSE(\sigma_n^*)$ obtained by *BSP* is smaller than the one obtained by *MBB*. Here σ_n^* is the bootstrapped value of the true value σ_n. This difference especially in the case of small sample ($n = 27$). This arise because the joins between blocks (in *MBB* method) there is independence between successive observations which does not occur in the original data. But in the *BSP* method, the dependence structure presented in the data is reproduced in bootstrap sample (see Mouiha and Rais, 2000, for more details).

Note that the executing time of the same application by using Borland Pascal is faster than the one used in S-Plus. The following results for the two models given above are for showing the fastness of Borland Pascal language:

Sample size	Model	Borland Pascal	S-Plus
n=27	M1	00h 12mn 27s 62cs	00h 14mn 35s 89cs
n=512	M1	03h 47mn 42s 31cs	04h 11mn 15s 75cs
n=27	M2	00h 16mn 21s 52cs	00h 18mn 25s 42cs
n=512	M2	03h 58mn 51s 68cs	04h 34mn 41s 93cs

Table 3.2: Executing time by S-Plus and Borland Pascal.

References

[1] Brockwell, P. J. and Davis, R. A. *Time Series: Theory and Methods*. New York: Spring Verlag, 1991.

[2] Davison, A. C. and Hinkley, D. V. *Bootstrap and their Application*. Cambridge: Cambridge University Press, 1966.

[3] Efron, B. Bootstrap Methods: another look at the jackknife. *Annals of Statistics 7* (1979), 1-26.

[4] Künsch, H. R. The jackknife and the bootstrap for general stationary observations. *Annals of Statistics 17* (1989), 1217-1241.

[5] Mouiha, A. and Rais, N. Bootstrapping the arithmetic mean for dependent observations. *Student vol 3* (2000), 273-279.