Studies in Nonlinear Dynamics and Econometrics

Quarterly Journal
July 1997, Volume 2, Number 2
The MIT Press

Studies in Nonlinear Dynamics and Econometrics (ISSN 1081-1826) is a quarterly journal published electronically on the Internet by The MIT Press, Cambridge, Massachusetts, 02142. Subscriptions and address changes should be addressed to MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142; tel.: (617) 253-2889; fax: (617) 577-1545; e-mail: journals-orders@mit.edu. Subscription rates are: Individuals $40.00, Institutions $130.00. Canadians add additional 7% GST. Prices subject to change without notice.

A Fast Algorithm for the BDS Statistic

Blake LeBaron
Department of Economics
University of Wisconsin, Madison
blebaron@facstaff.wisc.edu

Abstract. *The BDS statistic has proved to be one of several useful nonlinear diagnostics. It has been shown to have good power against many nonlinear alternatives, and its asymptotic properties as a residual diagnostic are well understood. Furthermore, extensive Monte Carlo results have proved it useful in relatively small samples. However, the BDS test is not trivial to calculate, and is even more difficult to deal with if one wants the speed necessary to make bootstrap resampling feasible. This short paper presents a fast algorithm for the BDS statistic, and outlines how these speed improvements are achieved. Source code in the C programming language is included.*

Keywords. specification testing, nonlinearity, chaos, BDS

1 Introduction

Early research into time-series analysis of chaotic systems led to several key insights. Most important was the fact that the standard toolbox of linear diagnostics needed to be extended, since many of these tests could be easily fooled into seeing randomness where rich nonlinear structures existed. The impact of these findings for modern time-series analysis has been significant, and most toolboxes now in use contain several different diagnostics for detecting potential nonlinearities.[1]

The BDS test, which was introduced in working paper form in the mid-1980s and published in Brock, Dechert, Scheinkman, and LeBaron (1996), has proved a very useful test in terms of size and power, and has been used in many nonlinear time-series studies. Also, extensive Monte-Carlo studies carried out in Brock, Hsieh, and LeBaron (1991) have shown that its small sample properties are quite good. Despite all these favorable features, the BDS test does have one drawback: it is not a trivial statistic to calculate. Easy-to-use code written for DOS-based computers was provided by W. D. Dechert, and later C code was provided by B. LeBaron. The Dechert code is very user-friendly, whereas the C source code uses a more complicated, but faster, algorithm. Also, as a C subroutine, it can be worked into the user's code, and extended for other purposes. This paper describes the algorithms used in this C code version.

The goals of this paper are to describe the BDS algorithms for the first time. It is hoped that this will both help new users, and also encourage those who would like to enhance, extend, and incorporate some of these methods in their own programs. The paper is in four short sections. After this introduction, the second section defines the BDS test and its asymptotic distribution properties. The third goes through the BDS algorithm, and the fourth concludes and suggests various extensions and modifications.

[1] Barnett et al. (forthcoming) provide a useful summary and comparison of many of these tests.

2 The BDS Test

The BDS test, which was inspired by algorithms used to estimate certain chaotic invariants, is developed in Brock et al. (1996), and extensively tested in Brock, Hsieh, and LeBaron (1991). It is a test for independence in a time series, but it can, and often is, used as a diagnostic for residuals of some conjectured model.

The basic intuition for the test is simple. Consider X_t to be a random series of length n. Specifically, let X_t be a univariate time series, independent, identically distributed from some distribution, F. Also, let

$$\begin{aligned} I_\epsilon(x, y) &= 1 \text{ if } |x - y| < \epsilon \\ &= 0 \text{ otherwise} \end{aligned}$$

Now define the event A as two points in X_t being within a distance ϵ of each other. In this notation, the probability of event A can be written as

$$P_A = E(I_\epsilon(X_t, X_s))$$

where the expectation is taken over F. Further define the event B as two points along with their predecessors in X_t being close. In other words, vectors of length 2 are both within ϵ of each other. This can be written

$$P_B = E(I_\epsilon(X_t, X_s) I_\epsilon(X_{t-1}, X_{s-1}))$$

Under independence of X_t, it is obvious that the two events contained in the compound event B are independent, and therefore $P_B = P_A^2$. This simple relation from probability is the basis of the BDS test. Through simple counting, one can estimate P_A and P_B, and also $P_B - P_A^2$, which under the null hypothesis has an expected value of zero. Brock et al. (1996) develop an asymptotic distribution theory for statistics of this form. More specifically, define

$$c_{m,n}(\epsilon) = \frac{2}{n(n-1)} \sum_{s=1}^{n} \sum_{t=s+1}^{n} \prod_{j=0}^{m-1} I_\epsilon(X_{s-j}, X_{t-j})$$

which estimates the probability that two m length vectors are within ϵ. Under the independent null hypothesis, it is clear that

$$E(c_{m,n}(\epsilon)) = (E(c_{1,n}(\epsilon)))^m$$

It is shown in (Brock et al. 1996) that

$$w_{m,n}(\epsilon) = \sqrt{n} \frac{c_{m,n}(\epsilon) - c_{1,n}^m(\epsilon)}{\sigma_{m,n}(\epsilon)}$$

has a limiting standard normal distribution with mean zero and unit variance, and $\sigma_{m,n}^2(\epsilon)$ is consistently estimated by

$$\sigma_{m,n}^2(\epsilon) = 4 \left[k^m + 2 \sum_{j=1}^{m-1} k^{m-j} c^{2j} + (m-1)^2 c^{2m} - m^2 k c^{2m-2} \right]$$

where $c = E(c_{1,n}(\epsilon))$, and $k = \int (F(z+\epsilon) - F(z-\epsilon))^2$. c is consistently estimated by $c_{1,n}(\epsilon)$, and k can be estimated by

$$k_n(\epsilon) = \frac{6}{n(n-1)(n-2)} \sum_{t=1}^{n} \sum_{s=t+1}^{n} \sum_{r=s+1}^{n} h_\epsilon(X_t, X_s, X_r)$$

$$h_\epsilon(i, j, k) = \frac{1}{3}(I_\epsilon(i, j)I(j, k) + I_\epsilon(i, k)I_\epsilon(k, j) + I_\epsilon(j, i)I_\epsilon(i, k))$$

At this point, the statistic is ready to be estimated. The above formulas are not that complicated, even though they are a little messy. They are nothing more than laborious counting, which computers should be good at. The problem is that they ask for so much counting that the computational time may get very large as n gets very large. Note that the k estimator is an order n^3 formula, which will blow up quickly, and the c estimator is order n^2, which can also become burdensome. The next sections will discuss some of the improvements that have been implemented to speed up estimation of the BDS statistic.

3 The Algorithm

Several authors have contributed improvements that have led to the faster estimation of BDS statistics. All of these improvements are included in the C version of the BDS algorithms included with this paper. This section will move from the simpler to the more complicated of these additions.

The first improvement was based on a suggestion of Theiler (Theiler 1990), not for BDS estimation, but to be used in dimension-estimation problems. Theiler recommends that for estimation of $c_{1,n}(\epsilon)$, the X_t series should be sorted first. Rather than searching through the entire series for the neighbors near to X_s, a sorted table is used so that it is only necessary to search a short distance in either direction. Once $X_t - X_s > \epsilon$, search can stop in this direction. This method can be applied to estimate $c_{m,n}(\epsilon)$ for $m > 1$, but this would require building and sorting a new table each time. In the BDS software it is used in the estimation of $c_{1,n}(\epsilon)$ and k only.

The second improvement was a more efficient estimation of k, and was discovered by W. D. Dechert. Dechert shifted to writing some of the above formulas, which are in the class of U statistics, as V statistics. The only difference between U and V statistics is that V statistics count the distance from point j to itself, which is less than ϵ, and gives a 1 for the I_ϵ indicator function. In terms of V statistics, expressions for c and k become

$$c^V_{m,n}(\epsilon) = \frac{1}{n^2}\sum_{s=1}^{n}\sum_{t=1}^{n}\prod_{j=0}^{m-1} I_\epsilon(X_{s-j}, X_{t-j})$$

$$k^V_n(\epsilon) = \frac{1}{n^3}\sum_{t=1}^{n}\sum_{s=1}^{n}\sum_{r=1}^{n} I_\epsilon(X_t, X_s)I_\epsilon(X_s, X_r)$$

Viewing the estimators in this format showed that k could be written as

$$k^V_n(\epsilon) = \frac{1}{n^3}\sum_{t=1}^{m}(\sum_{s=1}^{n} I_\epsilon(X_t, X_s))^2$$

which makes calculating k an order n^2 operation. It is easy to readjust the V statistics back to U statistics, and the C program does this for some of its estimates.

The final improvement to the algorithm is used in estimating $c_{m,n}(\epsilon)$ itself. This is still an n^2 algorithm, but it can be made more manageable by noticing that many calculations can be made in parallel, and much of the information from $m-1$ can be stored in a way that makes calculating $c_{m,n}(\epsilon)$ fairly easy.

The first step is to create an array, B, corresponding to whether each pair of points is within ϵ. In other words, entry $B_{i,j}$ will be 1 if $|x_i - x_j| < \epsilon$, and zero otherwise. Also, note that from this definition it is easy to see that the matrix, B, is symmetric, and has ones on the diagonal. It is therefore only necessary to store half

of this matrix. An example might look like this:

$$\begin{array}{ccccc} 1 & 1 & 1 & 0 & 1 \\ \vdots & 1 & 0 & 0 & 1 \\ \vdots & \vdots & 1 & 0 & 1 \\ \vdots & \vdots & \vdots & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & 1 \end{array}$$

Now imagine the entire array shifted to the left, moving the blank spaces to the right:

$$\begin{array}{ccccc} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & \vdots \\ 1 & 0 & 1 & \vdots & \vdots \\ 1 & 0 & \vdots & \vdots & \vdots \\ 1 & \vdots & \vdots & \vdots & \vdots \end{array}$$

To find the counts to calculate $c_{2,m}(\epsilon)$, we only need to look at $B_{i,j}$ and the position directly below it, since that position will correspond to whether $|x_{i+1} - x_{j+1}| < \epsilon$. If both positions are 1, then $B_{i,j}$ is replaced by a 1, if only one of the positions is 1, or both are 0, then $B_{i,j}$ is set to 0. This is a simple **and** operation performed on these two table entries, which are vertically adjacent. There is no reason that this couldn't be performed in parallel. Bitwise **and** each row with the row below it, replacing the row with that result. Move down through all rows in this fashion.

The only tricky part comes at the end of the row, where the bits have no corresponding entry below them. This refers to points for which the dimension m has run off the end of the series. For example, $m = 3$ for the point $i = n - 1$. It is assumed in the program that these points are set to zero. This would be equivalent to filling the matrix with zeros from the right of the diagonal in the above pictures.

The final complication comes in actually implementing this on the computer. The computer sets up the above table as a packed array of 15 bits/word. The shift is implicit in the table entries so that the procedure described above involves a simple **and** operation performed on multiple words. The final problem comes in counting the bits. This is done using a table lookup. A table of length 2^{15} is created, and each element stores the number of bits set to 1 in that entry's index. This table is then used to add up the number of bits set in the table, which is much faster than raw counting.[2]

4 Using the Software

The BDS software is designed with two users in mind: the casual user who needs a quick and easy way to calculate the diagnostic statistic, and the sophisticated user who wants to get in and modify the code, and use the subroutines along with other procedures of his or her own design. To accomplish this goal, most of the calculations for the BDS statistic are done in several major subroutines, allowing the user access to call these from his or her own programs. Also included is a command-line "front end," that implements the BDS test as a UNIX command, and can be given at the terminal or in batch files. The operation of the command-line interface is detailed first. The program is executed using the following command line.

[2]This setup still owes some connections to the fact that many machines had only 16-bit integers in the days it was written. Moving to 32-bit representations would be useful, but the table lookup would still be a little unwieldy on most modern machines. The entry in the table with binary address 000000000001010 would be 2, the number of bits set to 1.

bdstest filename maxdim epsilon

The parameters are:

1. filename: File name containing the time series in a single column separated by carriage returns.
2. maxdim: The maximum dimension, m, at which the statistic will be estimated. The program will print BDS statistics in the range of $j = 2, \ldots, m$. Also, the time-series length will be reduced by a factor of $m - 1$. For example, for a series of length 10, for $j = 2$, the test statistic can only be estimated over a range of x_1 through x_9. To make the estimates consistent across tests, the series is stopped at N, less the maximum dimension plus 1, $N - m + 1$.
3. epsilon: The epsilon parameter is given to the program in units of the time series. Often in practice this is set to a fraction of the standard deviation of the time series.

The output of the program is a simple vector corresponding to

$$w_2, w_3, \ldots, w_m,$$

the normalized statistics which are asymptotically distributed normal with zero mean and unit variance. The entire input/output system is set up, so the program can easily pipe its output to a file or another program on a UNIX machine.

To keep this front end extremely simple, several choices about how to implement the BDS test have been made. These should not be taken as hard rules, and the user should view these as flexible parts that can be changed. Specifically, steps 2 and 3 above can be changed. The user may want the program to use the maximum amount of data for each different dimension, moving the endpoint backwards as the dimension is increased. Also, the user may want to automatically scale the data by its standard deviation.

5 Modifying the code

Experienced C programmers may want to modify the code, or use the subroutines with their own programs. This section outlines their structure.

The main program handles simple inputs from the user, and reading the data. Most of the serious work is handled by two main subroutines. Their calling sequences follow.

void fkc(x,n,k,c1,c,m,mask,eps)

1. x: Vector of series to test (double *), but it can be modified using the PREC definition set at the beginning of the program. Setting PREC to *float* or *int* will allow the use of other types of series.
2. n: length of series (int).
3. k: returned value of k (double *) (V statistic estimate).
4. c1: returned value of c (double *) (V statistic estimate).
5. c: returned raw c values $c[1]$, $c[2]$, $c[3]\ldots$. (double *) (U statistic estimate).
6. m: maximum embedding—BDS statistics will be calculated for $i = 2$ to m (int).
7. mask: number of points to ignore at the end of the series. Since the calculation of $c_{2,n}$ can effectively use more points than $c_{3,n}$, $c_{4,n}$, ..., often the last several points are ignored so that all statistics are calculated on the same set of points. For $m = 3$, we might only use x_1 through x_{n-2} for the calculations of $c_{2,n}$ and $c_{3,n}$. This is generally set to $m - 1$ to allow all c to be estimated on a point set of $n - m + 1$ (int).

Table 1
Estimated Time Required to Run BDS using Fast and Slow Algorithms.

N	Fast	Slow
1,000	0.33	2.00
2,500	1.95	12.47
5,000	7.98	48.89
10,000	32.97	203.00

Note: Times are all given in seconds. *N* refers to the time-series length. *Fast* is the algorithm described here, and *Slow* refers to a slower algorithm using brute-force counting (which is also distributed with the BDS code, as the routine fkc_slow).

8. eps: epsilon value for close points (double), or set to (PREC).

double cstat(c,cm,k,c1,m,n)

1. c: $c[1]$ c for embedding 1 (value returned by fkc) (U statistic estimate) (double).
2. cm: $c[m]$ c for embedding m (U statistic estimate) (double).
3. k: k estimate (returned by fkc) (V statistic) (double).
4. c1: c for embedding 1 (returned by fkc, $c1$) (V statistic) (double).
5. m: embedding (int).
6. n: length of series (int).
7. returns w_m normalized BDS statistic (double).

5.1 A note on memory usage

The gains in speed in the BDS algorithm come from the storage of results from previous embeddings. Therefore, the memory usage of the program is quite large. The program automatically allocates the space it needs and checks if it is available before running. An error message will indicate that all space has been allocated.

A quick estimate of the kind of space necessary can be obtained by estimating the size of the grid matrix needed. First, the size of the matrix in bits is $n(n-1)/2$, since it is an upper triangular matrix and the diagonal is not needed since it is all ones. This can then be divided by 8 to give the approximate size in bytes. For example, a series of length 10,000 would require approximately 6.2 megabytes.[3]

5.2 Run times

To give users an idea of the time required to run the BDS software, some short tests were performed using the fast algorithm described here, and a slower version that uses a slower, more direct approach of counting pairs. The results are given in Table 1. The run times are from a Pentium 90, running NeXTstep 3.3 with 32 megabytes of memory.

[3]This estimate depends on the ability to use 16-bit integers.

6 Extensions and Conclusions

The BDS software is now is almost 10 years old. It has been distributed widely, and has been a reliable tool for estimating statistics of this form. However, there are many directions in which this code can be modified. Among these might be clever speedups, and a nicer user interface. Also, a link into major software packages is another possibility.[4] Finally, other statistics can be estimated with small modifications of this code.[5] It is hoped that this software will prove useful for others in many different fields.

References

Barnett, W. A., A. R. Gallant, M. J. Hinich, J. A. Jungeilges, D. T. Kaplan, and M. J. Jensen (forthcoming). "A Single-Blind Controlled Competition among Tests for Nonlinearity and Chaos." *Journal of Econometrics*.

Brock, W. A., W. D. Dechert, J. A. Scheinkman, and B. LeBaron (1996). "A Test for Independence Based on the Correlation Dimension." *Econometric Reviews* 15(3):197–235.

W. A. Brock, D. Hsieh, and B. LeBaron (1991). *Nonlinear Dynamics, Chaos, and Instability: Statistical Theory and Economic Evidence*. Cambridge, MA: MIT Press.

Grassberger, P., and I. Procaccia (1982). "Characterization of Strange Attractors." *Physical Review Letters* 50:346–349.

Scheinkman, J. A., and B. LeBaron (1989). "Nonlinear Dynamics and Stock Returns." *Journal of Business* 62:311–338.

Theiler, J. (1990). "Estimating Fractal Dimension." *Journal of the Optical Society of America A* 7:1055–1073.

[4]The author is presently considering a link to Matlab as a .mex extension.

[5]Examples include the *S* statistics estimated in (Scheinkman and LeBaron 1989), or the original dimension estimates in (Grassberger and Procaccia 1982). For the latter, this might be a somewhat inefficient program, since dimension estimation requires estimates at many different values of ϵ, and this program is most efficient at one ϵ and many dimensions.

Editorial Policy

The *SNDE* is formed in recognition that advances in statistics and dynamical systems theory may increase our understanding of economic and financial markets. The journal will seek both theoretical and applied papers that characterize and motivate nonlinear phenomena. Researchers will be encouraged to assist replication of empirical results by providing copies of data and programs online. Algorithms and rapid communications will also be published.

ISSN 1081-1826