ISSN 1330–0016
CODEN FIZBE7

LETTER TO THE EDITOR

FAST PROCEDURE FOR ESTIMATING CAPACITY DIMENSION OF THE FRACTAL OBJECTS BY THE BOX COUNTING

ALEKSANDAR MAKSIMOVIĆ, STJEPAN LUGOMER and BOŽIDAR VOJNOVIĆ

Rugjer Bošković Institute, Bijenička c. 54, 41000 Zagreb, Croatia

Received 15 February 1995

UDC 531.19

PACS 02.70.-c, 05.20.-y

A fast procedure for computing fractal dimension by the box counting is derived. An approximation for the number of visited boxes $N_B(\epsilon, n)$ as a function of n is given. The memory requirement is calculated with this function and the scaling for derivative $\Delta N(\epsilon, n)/\Delta n \approx \text{const} \times \beta\epsilon^{-\alpha} n^{-\beta-1}$ given by Grassberger is obtained for the non saturated segment. The procedure is tested on the calculation of fractal dimension of *Sierpinski triangle* and the *Hénon* map, and compared with the procedure of Grassberger.

A fractal objects like strange attractors are typically characterized by *fractal dimension*. The fractal dimension D of a set A can be obtained from the relation

$$N(A, \epsilon) \approx C\epsilon^{-D} \tag{1}$$

where $N(A, \epsilon)$ is the number of cubes with the edge size ϵ that contain a piece of the fractal object and C is a positive constant [1,3]. From Eq. (1), a dimension D

(usually called capacity) can be obtained [1,4,5]:

$$D = \lim_{\epsilon \to 0} \frac{\log N(A, \epsilon)}{\log(1/\epsilon)}. \tag{2}$$

In the so called box–counting algorithm one counts the minimal number of boxes—cubes $N_{\mathrm{B}}(\epsilon)$ that cover the set with boxes of the size ϵ [2].

This letter gives description of the faster procedure for calculation of the fractal dimension than the one given in Ref. 2. The memory usage is estimated by using the best fitting function in the saturated region. From this function we obtain the same scaling for the derivative of the number of visited boxes, as given by Grassberger [4].

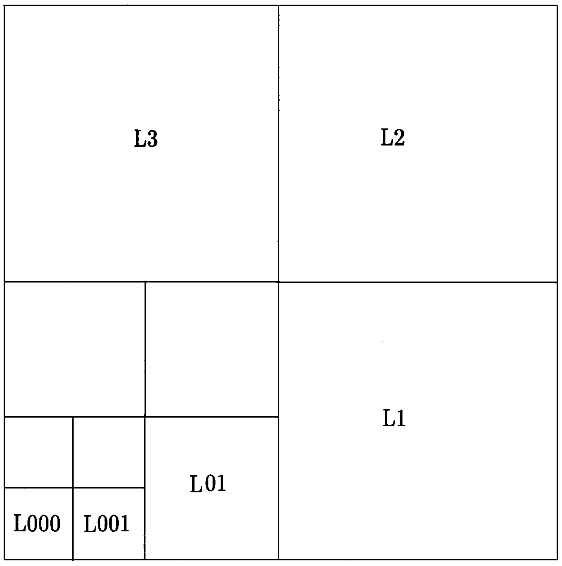

Fig. 1. The first three subdivisions of the box and labels which are used in the new procedure.

Each of n points of a set embedded in d_{e} dimensions can be represented by a vector with coordinates $\{\mathbf{X}_i;\ i = 1, d_{\mathrm{e}}\}$. The values of $\mathbf{X}_i$ are normalized to cover the range $(0, 2^k - 1)$ and the set is covered by the grid of d_{e} dimensional cubes of edge size 2^{k-m}, where $m = 0, \ldots, k$. The procedure is implemented for embedding dimension $d_{\mathrm{e}} = 2$. Every normalized vector $\mathbf{X}_i$ is represented by k–bits $\mathbf{X}_i = (\sigma_1^i, \ldots, \sigma_k^i)$. Figure 1 shows the first three divisions of a box. For each vector $\mathbf{X_i}$ we strip the first bit σ_1^i. In $d_{\mathrm{e}} = 2$ dimensions, combinations of pairs of bits (σ_1^1, σ_1^2) determine which box covers this point. We adopt following combinations $(\sigma_1^1, \sigma_1^2) = \{\{0,0\}, \{0,1\}, \{1,0\}, \{1,1\}\}$ which correspond to the boxes $\{L_0, L_1, L_2, L_3\}$ in Fig. 1. This is easily generalized to d_{e} embedding dimensions replacing the pairs of bits with d_e–dimensional vector of bits $(\sigma_1^1, \ldots, \sigma_1^{d_e})$. In the next step we strip second bit from the vector and repeat the above procedure by looking at the pairs of them and determine which box is occupied in the next subdivision (i.e box L_{01} in Fig. 1). This procedure is repeated for different boxes of the edge size 2^{k-m}, where $m = 0, \ldots, k$. The first m bits from vector $\mathbf{X}_i$ determines in which box of size 2^{k-m} they belong.

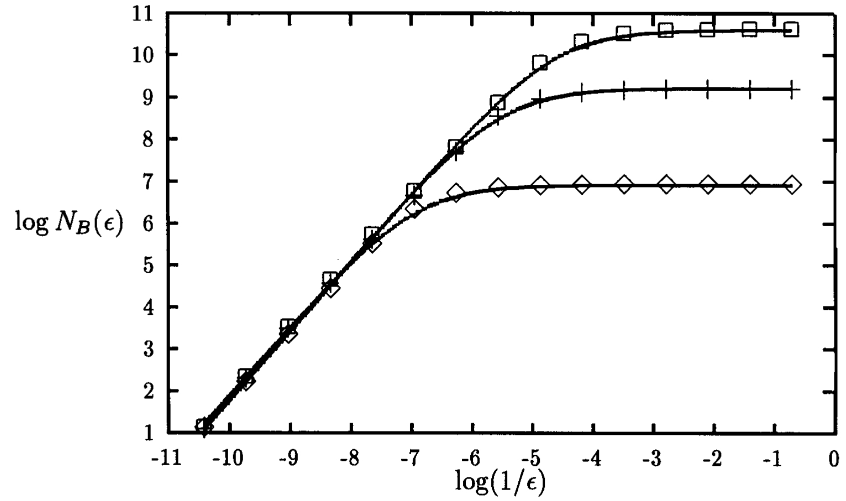

Fig. 2. Plot of the logarithm of the numbers $N_{\rm B}(\epsilon; n)$ of non-empty boxes versus $\log(1/\epsilon)$ for Sierpinski triangle. $\diamond$: n =1 000; +: n =10 000; $\square$: n =40 000.

We start with one root node with four pointers, which represent four boxes in the first division $\{L_0, L_1, L_2, L_3\}$, of the size 2^{k-1}. If a pointer contains special value "**NULL**" there are no points in that box; otherwise it points to the structure of the four pointers, each of them representing further subdivision of that box. Using the first m bits of a number $\mathbf{X}$ as described above, we get m pointers, every one of them showing which of the boxes is occupied in the m-th subdivision. This is repeated for all members in the data set. In this way only one scan of the data set is enough to remember all positions, and the one walk through the tree gives the number of the visited boxes. The algorithm was performed in a *C++* program.

The original algorithm taken from Ref. 2 was also performed in a *C++* program in order to compare the execution times of both programs. Here is the brief description of the algorithm.

By using a bitwise operation **AND** and the mask M we can strip the first m bits from x or y coordinate (which have the first m bits set to 1 and all others to 0). Thus the masking operation $\{x, y\}\&M$ gets the first m bits from vector $\{x, y\}$ which now have the form $\{(\sigma_1^1, \ldots, \sigma_m^1), (\sigma_1^2, \ldots, \sigma_m^2)\}$. Shifting the x coordinate by 16 bits and performing a bitwise **OR** operation we can form a number $z_n = (\sigma_1^1, \ldots, \sigma_m^1, \sigma_1^2, \ldots, \sigma_m^2)$ for each point in a data set, where $m = 0, \ldots, k$. Distinct

z_n corresponds to points in different boxes of the edge size 2^{k-m}. Using a heapsort algorithm we sort all z_n and then count the number of distinct z_n [6]. *This is essentially the algorithm described in Ref. 2, only the operation on strings is replaced with the equivalent operation on bits which increases the speed of the computation significantly.*

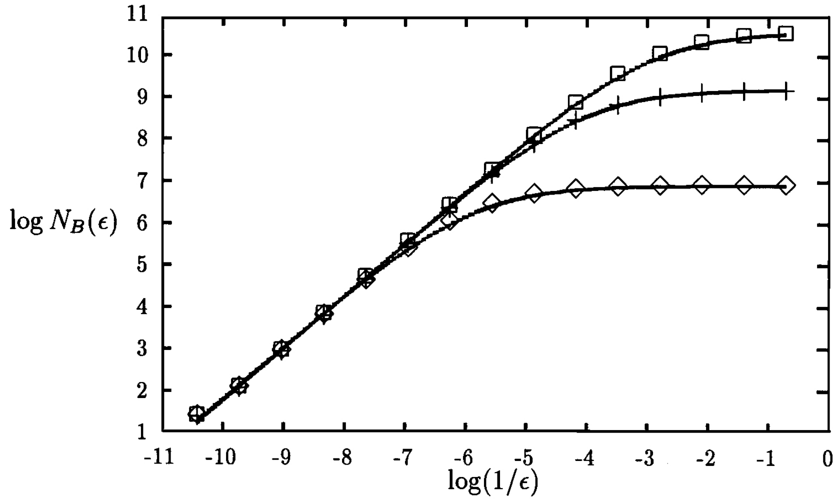

Fig. 3. Plot of the logarithm of the numbers $N_{\rm B}(\epsilon; n)$ of non-empty boxes versus $\log(1/\epsilon)$ *for Hénon attractor.* $\diamond$: *n =1 000;* +: *n =10 000;* $\square$: *n =40 000.*

To compare the accuracy and the computational times for the box counting and for our new procedure, we computed the fractal dimension $d_{\rm B}$ with the above algorithm for the two types of attractors. The first is a celebrated Hénon map [2]:

$$x_{i+1} = 1 + y_i + ax_i, \quad y_{i+1} = bx_i \tag{3}$$

where $a = -1.4$ and $b = 0.3$. The fractal dimension for this map is known from estimates based on the capacity dimension [2] and the correlation dimension [3]. The second type of fractal used is *Sierpinski triangle* obtained from the *Iterated function system*, and generated by using *Random iteration algorithm* [1]. The fractal dimension of the *Sierpinski triangle* is $\log(3)/\log(2)$. It is obtained by using the following set of a three affine transformations:

$$x_{i+1} = x_i/2 + a_k, \quad y_{i+1} = y_i/2 + b_k \tag{4}$$

with (k =1,2,3), where $a_1 = a_2 = b_1 = 1$, $a_3 = b_2 = b_3 = 50$, and all transformations were applied with equal probability [1,2].

When the number of filled boxes $N_{\rm B}$ is equal to the length of data set n so that all points lie in distinct boxes, the function is saturated. This part of the function

$N_{\rm B}$ requires the largest part of computer memory, and values near saturation should be disregarded when the slope of $\log N_{\rm B}$ versus $\log(1/\epsilon)$ is calculated [2].

The number of visited boxes $N_{\rm B}(\epsilon, n)$ can be approximated with the function

$$N_{\rm B}(\epsilon_k; n; \epsilon_{ko}) = n \frac{1}{(\epsilon_k/\epsilon_{ko})^{\alpha} + 1}, \tag{5}$$

where ϵ_k is the length of the box, and ϵ_{ko} is the constant for a given n. We find that the values of the exponent $\alpha_S = 1.65 \pm 0.2$ for the *Sierpinski triangle* and $\alpha_H = 1.251 \pm 0.003$ for the *Hénon* yield the best fit to the data for 14 different lengths of n between 1 000–40 000. Figure 2 shows a log–log plot of the number of visited boxes and the function $N_{\rm B}(\epsilon_k; n; \epsilon_{ko})$ versus $\log(1/\epsilon_k)$ for the *Sierpinski triangle*, and for the three different lengths of data set. This is also illustrated in Fig. 3 for the *Hénon* attractor.

However, ϵ_{ko} is a function of the length n, and in the log–log plot it behaves as a linear function of n, meaning that we approximate ϵ_{ko} with the power law:

$$\epsilon_{ko} = 1/Cn^{\gamma}, \tag{6}$$

where C is constant and γ is invariant for the given attractor, as it was suggested in Ref. 4. With the least square fit we obtain $\gamma_S = 0.635 \pm 0.002$ for the *Sierpinski triangle* and $\gamma_H = 0.810 \pm 0.001$ for the *Hénon* attractor (see Fig. 4). Inserting Eq. (6) in (5) one finds:

$$N_{\rm B}(\epsilon_k; n) = n \frac{1}{(\epsilon_k C n^{\gamma})^{\alpha} + 1}. \tag{7}$$

TABLE 1.
Constants α and β obtained by the fitting procedure.

Attractor	Sierpinski triangle	Hénon
α	1.65 ± 0.02	1.251 ± 0.03
γ	0.635 ± 0.002	0.810 ± 0.001
$\alpha\gamma$	1.056	1.008

TABLE 2.
Memory required by the program given in bytes for various values of the maximal range k used in rescaling.

Sierpinski tringle			Hénon attractor		
k–range	n	Memory usage	k–range	n	Memory usage
1–16	1 000	307 050	1–16	1000	258 720
1–16	10 000	2 441 140	1–16	10 000	1 724 840
1–16	40 000	8 250 000	1–16	40 000	4 855 000
1–12	10^6	3 155 900	1–12	10^6	5 600 000
1–12	10^7	5 025 600	1–12	10^7	5 700 000

TABLE 3.
The capacity dimension from Ref. 2 and the calculated one with the new procedure for two lengths of data set n.

	Capacity dimension from Ref 2		Calculated capacity dimension	
n	1 000	10 000	1 000	10 000
Hénon attractor	1.32 ± 0.02	1.27 ± 0.02	1.24 ± 0.02	1.245 ± 0.008
Sierpinski triangle			1.58 ± 0.01	1.56 ± 0.03

TABLE 4.
Speed of the program execution given in seconds for two lengths of data set n.

	algorithm from Ref. 2		New procedure	
n	1 000	10 000	1 000	10 000
Hénon attractor	5.00	36.80	3.19	15.43
Sierpinski triangle	4.78	37.02	3.30	16.70

Table 1 gives the summary for the values of constants α and γ for these two attractors. In the limit $(\epsilon_k C n^\gamma)^\alpha \gg 1$ Eq. (7) gives:

$$N_{\rm B} \approx n(\epsilon_k C n^\gamma)^{-\alpha}. \tag{8}$$

This equation has the same form as the expression for the fractal dimension (apart from the factor n^γ , which is constant if one data set is examined, i.e. n is constant), but the value of the constant α is always greater than fractal dimension because the saturated segment lowers the slope given by the non–saturated segment.

Obviously the substitution $\gamma\alpha - 1 = \beta$, gives the same scaling as given in Ref. 4 for derivative $\Delta N(\epsilon, n)/\Delta n \approx \text{const} \times \beta\epsilon^{-\alpha} n^{-\beta-1}$. Finally, we can estimate the memory required by the program. As pointed above, the program requires $\sum_k N_{\rm B}(\epsilon_k; n) \times$ size of Node bytes of memory, where the size of the Node is 32 bytes. The memory consumption is summarized in Table 2 for various values of the maximal set rescaling range k.

In order to compare the values of capacity dimension, we used the same conditions as in Ref. 2. Thus, only the values $N_{\rm B} \leq N/5$ were taken into account with the limitation that, because of poor resolution, $N_{\rm B}$ is ignored for $m = k, k-1$, where $k = 16$. We then used the least square fit to determine $d_{\rm B}$ as the slope of $\log N_{\rm B}$ versus $\log(1/\epsilon)$, as illustrated in Fig. 5 for the *Sierpinski triangle*, and for the *Hénon attracor* in Fig. 6. Comparison of capacity dimension obtained by the new procedure with the one from Ref. 2 is given in Table 3. It shows that agreement is satisfactory in general, while for N =10 000 points the agreement is in the limit of error. Computational times for the new procedure and the algorithm from Ref. 2 are given in Table 4.

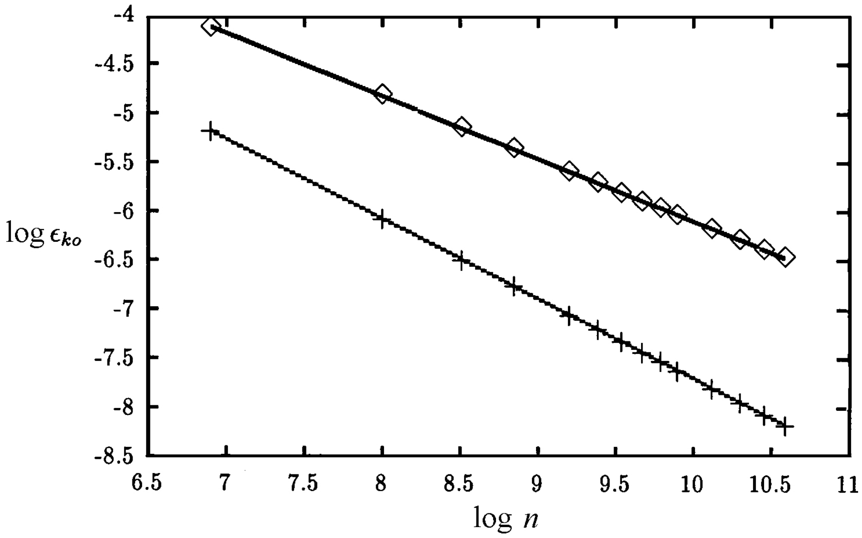

Fig. 4. Plot of the logarithm of the $\log \epsilon_{ko}$ *versus* $\log n$ *for Hénon attractor and Sierpinski triangle.* $\diamond$*: Sierpinski triangle;* $+$*: Hénon attractor.*

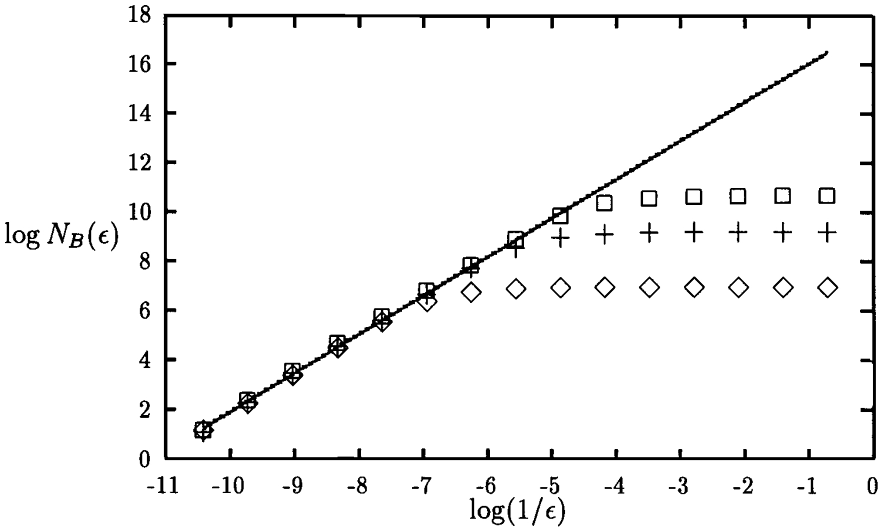

Fig. 5. Plot of $\log N(\epsilon)$ *versus* $\log(1/\epsilon)$ *for Sierpinski triangle with different length of data set N =1 000, 10 000, 40 000. Also shown is the least square fit as a full line.*

All calculations were done on PC386, while the programs were compiled with gnu compiler which exists on many platforms. Compiler supports the virtual memory, so a large data set can be processed by means of swapping data to the disk. With $k = 16$ bits used for rescaling maximal length of data set (assuming all points in set are distinct) is $\approx 65\,000$; for the larger data set we must use a larger k i.e. $N < 2^k$.

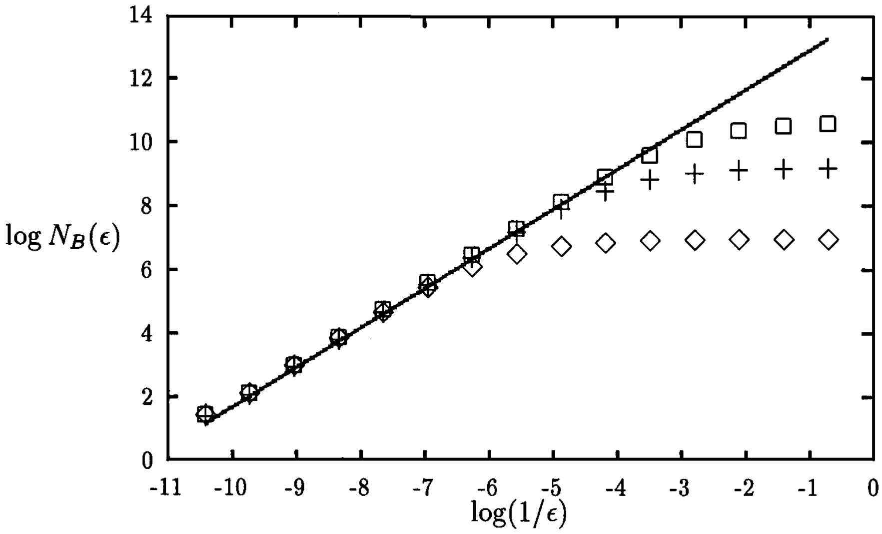

Fig. 6. Plot of $\log N(\epsilon)$ *versus* $\log(1/\epsilon)$ *for Hénon map with different length of data set N =1 000, 10 000, 40 000. Also shown is the least square fit as a full line.*

Acknowledgements

We would like to thank Nada Bosnić for very useful discussion. Eventual request for the program please send by E–mail to: maks@olimp.irb.hr.

References

1) M. Barnsley, *Fractals Everywhere* (Academic Press, New York, 1988);

2) L. S. Liebovitch and T. Toth, *Phys. Lett. A***141** (1989) 386;

3) P. Grassberger and I. Procaccia, *Phys. Rev. Lett.* **50** (1983) 346;

4) P. Grassberger, *Phys. Lett. A***97** (1983) 224;

5) A. Giorgilli, D. Casati, L. Sironi and L. Galgani, *Phys. Lett. A***115** (1986) 202;

6) W. H. Press, B. P. Flanerry, S. A. Teukolsky and W. T. Vetterline, *Numerical Recipes* (Cambridge Univ. Press, Cambridge, 1986).

BRZI POSTUPAK PROCJENE KAPACITIVNE DIMENZIJE FRAKTALNIH OBJEKATA METODOM BROJANJA KVADRATA

ALEKSANDAR MAKSIMOVIĆ, STJEPAN LUGOMER I BOŽIDAR VOJNOVIĆ

Institut Rugjer Bošković, Bijenička c. 54, 41000 Zagreb, Hrvatska

UDK 531.19

PACS 02.70.-c, 05.20.-y

Izložen je brzi postupak za računanje fraktalne dimenzije pomoću metode brojanja kvadrata (box counting). Dana je aproksimacija broja popunjenih kvadrata $N_B(\epsilon, n)$ kao funkcija od n. Tom funkcijom izračunati su memorijski zahtjevi i dobiveno je skaliranje Grassbergerove derivacije $\Delta N(\epsilon, n)/\Delta n \approx \text{konst} \times \beta\epsilon^{-\alpha}n^{-\beta-1}$ za nezasićeno područje. Postupak je provjeren na proračunu fraktalne dimenzije za *trokut Sierpinskog* i za *Hénonovo preslikavanje*, te je uspoređen sa Grassbergerovim postupkom.