

## A simpler and elegant algorithm for computing fractal dimension in higher dimensional state space

S GHORUI, A K DAS and N VENKATRAMANI

Laser and Plasma Technology Division, Power Plasma Devices Section, Bhabha Atomic Research Centre, Mumbai 400 085, India

MS received 26 November 1999

**Abstract.** Chaotic systems are now frequently encountered in almost all branches of sciences. Dimension of such systems provides an important measure for easy characterization of dynamics of the systems. Conventional algorithms for computing dimension of such systems in higher dimensional state space face an unavoidable problem of enormous storage requirement. Here we present an algorithm, which uses a simple but very powerful technique and faces no problem in computing dimension in higher dimensional state space. The unique indexing technique of hypercubes, used in this algorithm, provides a clever means to drastically reduce the requirement of storage. It is shown that theoretically this algorithm faces no problem in computing capacity dimension in any dimension of the embedding state space as far as the actual dimension of the attractor is finite. Unlike the existing algorithms, memory requirement offered by this algorithm depends only on the actual dimension of the attractor and has no explicit dependence on the number of data points considered.

**Keywords.** Dimension; chaos; fractal; attractor.

**PACS Nos** 05.45.pq; 05.45.tp; 05.45ac; 05.45.df

### 1. Introduction

The dimension of a dynamical system is the minimum number of state variables required to describe the dynamics of the system. In differential topology, dimension of a manifold is same as the dimension of the Euclidean space that the manifold resembles locally. All these dimensions are integers. However, chaotic attractors, observed widely in various physical systems, fall in a very special category. Almost all of them exhibit fractal dimensions. There are a number of statistical measures for dimension such as, capacity dimension ( $D_{\text{cap}}$ ), information dimension ( $D_I$ ), correlation dimension ( $D_C$ ) etc., all of which says something about the amount of information necessary to characterize the system. Renyi [1] defined an entire family of dimensions which includes  $D_{\text{cap}}$ ,  $D_I$ ,  $D_C$  as special cases as

$$D(q) = \text{Lt}_{\varepsilon \rightarrow 0} \frac{H_q(\varepsilon)}{\ln(1/\varepsilon)} \quad q = 0, 1, 2, \dots, \quad (1a)$$

where

$$H_q(\varepsilon) = \frac{1}{1-q} \ln \sum_{i=1}^{N(\varepsilon)} P_i^q(\varepsilon) \quad q \neq 1, \quad (1b)$$

$$H_q(\varepsilon) = - \ln \sum_{i=1}^{N(\varepsilon)} P_i(\varepsilon) \ln P_i(\varepsilon) \quad q = 1. \quad (1c)$$

The attractor is fully covered up by  $N(\varepsilon)$  hypercubes of dimension  $\varepsilon$ . (A hypercube is a line in one dimension, a square in two dimension, a cube in three dimension and so on).  $H_q$  is the generalized entropy and  $P_i$  is the relative frequency of visiting a typical trajectory to the  $i$ th volume element. For various values of  $q$ , various dimensions come out:  $D_{\text{cap}} = D(0)$ ,  $D_1 = D(1)$ ,  $D_C = D(2)$ , etc. [1]. However, for same number of data points and under identical conditions, computed capacity dimension for a system is much nearer to the actual dimension compared to that provided by others (see [2]). Nevertheless, a quick scan through literature will reveal that till now correlation dimension is more popular compared to capacity dimension. This is most probably because as far as computation is concerned, it is much easier to compute correlation dimension than capacity dimension. If there are a collection of  $N$  points of a trajectory either through simulation or from measurements, the problem of finding correlation dimension of the system ultimately reduces to finding the number, ( $N_p$ ), of pair of points  $(x_i, x_j)$ , such that

$$\|x_i - x_j\| < \varepsilon. \quad (2)$$

The equation for correlation dimension get reduced to

$$D_c = \text{Lt}_{N \rightarrow \infty} \frac{1}{N^2} [N_p]. \quad (3)$$

This way it avoids the requirement of enormous storage. However for computing capacity dimension, there is no other way except going for rigorous counting of the number of boxes filling the attractor. If  $L$  is the maximum extent of the attractor considering all directions in the state space of dimension  $d$ , the total number of boxes required to fill up the considered state space is,  $N_b(\varepsilon) = (L/\varepsilon)^d$ . For example if  $(L/\varepsilon) = 1000$ , which is very common, in two dimensional state space we need 10 lakh squares, in three dimensional state space we need  $10^9$  cubes and in 4 dimensional state space we need  $10^{12}$  hypercubes and so on. In conventional technique, usually all these boxes are initialized to zero at the starting. This needs a memory size of  $N_b(\varepsilon)$ . Then each point on the attractor is considered and determined to which box it goes. The corresponding box is then labeled 'one'. When all points on the attractor are covered, total number of boxes with mark 'one' are determined by checking all the  $N_b(\varepsilon)$  memory locations and the capacity dimension is computed using formula 1. So it is evident that if further smaller volume elements are taken and the computation is to be done in higher dimensional state space, the memory requirement becomes so huge that the computation becomes too difficult if not impossible.

In this decade a number of attempts has been made to come out from this difficulty. In [3], Liebovitch and Toth devised an algorithm in which they re-scaled the points in  $d$ -dimensional embedding state space in the interval  $(0, 2^k - 1)$  and represented each in binary form. Then they covered the attractor using  $d$ -dimensional boxes of edge size  $2^{-m}$  ( $0 \leq$

$m \leq k$ ) and checking the most significant  $m$  bits of each point, to which box the point goes was determined. To determine the number of boxes needed to cover the attractor, they masked the least significant  $(k-m)$  bits of each point with zeros, sorted the list using optimal shorting procedure (quick short or heapsort [4]) and then determined the distinct values of the masked points. This requires a memory size of  $d.O(N)$  but execution time is increased to  $O(N * \ln(N))$ . In [5], Hou, Gilmore, Mindlin and Solari introduced a new topological ordering, in which they intercalated the coordinates of each point into a long bit string and sorted the points according to the value of this string. Only one shorting was needed in this algorithm. It reduced the execution time but memory requirement was same as earlier. Block, Bloh and Schellnhuber [2], improved this algorithm by counting the boxes in a special economic way and made the consumed computer memory and time both quasi-proportionate to the size of input data set. Later on Molteno [6] introduced the successive partitioning algorithm where the whole data set was initially covered by a  $d$ -dimensional box and the box was allowed to get partitioned recursively and evenly up to a predetermined extent. In each step the boxes carrying no point were removed from the calculation. It reduced the computation time and storage both to the order of  $N$ . The algorithm we are presenting requires a memory which does not depend on  $N$ . But it depends on  $N(\varepsilon)$ , the number of filled boxes which depends only on the actual dimension of the attractor. It needs a computation time proportional to  $O(N)$  and faces no problem in computation as far as the actual dimension of the attractor is finite.

## 2. The algorithm

The present algorithm is originated from the idea that, although  $N_b(\varepsilon)$  becomes enormously large for a given small value of  $\varepsilon$ ,  $N(\varepsilon)$ , the number of filled boxes, never become so. This becomes clear if we look at the explicit form of 1(a) for capacity dimension:

$$D_{\text{cap}} = \lim_{\varepsilon \rightarrow 0} \frac{\ln[N(\varepsilon)]}{\ln[1/\varepsilon]}. \quad (4)$$

If  $\varepsilon$  is finite, for a practical system  $N(\varepsilon)$  is totally determined by  $D_{\text{cap}}$ , which is finite. Under various  $d$  and  $\varepsilon$ , a comparison between  $N(\varepsilon)$  and  $N_b(\varepsilon)$  is given in table 1 for henon attractor (mapped to the interval (0,1) using affine mapping), for which  $D_{\text{cap}} = 1.26$ . The huge difference in magnitude between  $N(\varepsilon)$  and  $N_b(\varepsilon)$  is explicitly noticeable for higher values of  $d$ . This observation is judiciously exploited by the present algorithm.

In our algorithm the data set in question is first mapped into the interval [0,1] using affine mapping which does not alter the dimensional properties of the data set anyway. All subsequent operations are done on this newly formed data set. This is necessary to make the code generally applicable to any kind of data set. Using embedding technique, attractor is then reconstructed in  $d$ -dimensional state space. Let the  $j$ th point on the attractor has coordinates  $(x_1, x_2, x_3, \dots, x_d)$ . The box to which this coordinate goes is marked by the unique indexing technique

$$Ib_j = k_1 + \sum_{i=2}^d (k_i - 1)M^{i-1}. \quad (5a)$$

Here

**Table 1.** Memory requirement statistics for henon attractor:  $X_{n+1} = 1 - aX_n^2 + Y_n, Y_{n+1} = bX_n$  ( $a = 1.4, b = 0.3$ ).  $N(\varepsilon)$  = storage required in our algorithm (the nearest upper side integer is listed),  $N_b(\varepsilon)$  = storage required by conventional technique,  $d$  = dimension of embedding state space.

$\varepsilon$	$N(\varepsilon) = D_{\text{cap}} * \ln(1/\varepsilon)$	$N_b(\varepsilon) = (1/\varepsilon)^d$						
		$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$
0.1	19	$10^2$	$10^3$	$10^4$	$10^5$	$10^6$	$10^7$	$10^8$
0.05	44	$4 \times 10^2$	$8 \times 10^3$	$1.6 \times 10^4$	$3.2 \times 10^5$	$6.4 \times 10^6$	$1.28 \times 10^8$	$2.56 \times 10^8$
0.01	332	$10^4$	$10^6$	$10^8$	$10^{10}$	$10^{12}$	$10^{14}$	$10^{16}$
0.005	793	$4 \times 10^4$	$8 \times 10^6$	$1.6 \times 10^9$	$3.2 \times 10^{10}$	$6.4 \times 10^{12}$	$1.28 \times 10^{15}$	$2.56 \times 10^{16}$
0.001	6025	$10^6$	$10^9$	$10^{12}$	$10^{15}$	$10^{18}$	$10^{21}$	$10^{24}$
0.0005	14432	$4 \times 10^6$	$8 \times 10^9$	$1.6 \times 10^{13}$	$3.2 \times 10^{15}$	$6.4 \times 10^{18}$	$1.28 \times 10^{22}$	$2.56 \times 10^{24}$
0.0001	109647	$10^8$	$10^{12}$	$10^{16}$	$10^{20}$	$10^{24}$	$10^{28}$	$10^{32}$

$$k_i = \text{Int} \left( \frac{x_i}{\varepsilon} \right) + 1, \tag{5b}$$

$$M = \text{Int} \left( \frac{1}{\varepsilon} \right) + 1. \tag{5c}$$

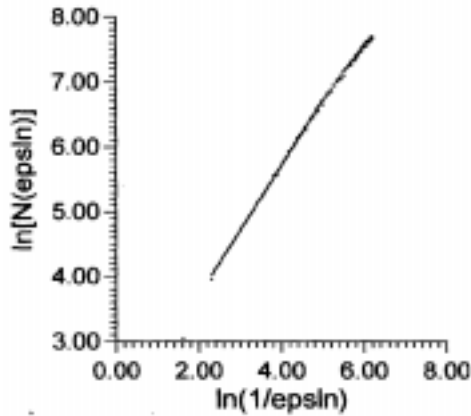
This indexing technique is unique in the sense that the combination of  $x_i$  representing a point uniquely determines the value of  $Ib_j$ . No other combination of  $x_i$  can result in same  $Ib_j$ . Only those values of  $Ib_j$ , which are distinct, are stored in computer memory. Obviously the maximum number of such distinct  $Ib_j$  values can never exceed  $N(\varepsilon)$ . This way it fixes the upper limit of memory requirement to  $N(\varepsilon)$ . The points on the attractor are scanned one by one and  $Ib_j$  corresponding to each point are compared with the already stored distinct values of  $Ib_j$ . If it differs from the already stored values, the filled box index (FBI) is increased by one and this value of  $Ib_j$  is included in the stored list of distinct  $Ib_j$  values. Otherwise FBI and list of stored  $Ib_j$  are left unchanged and next point on the attractor is examined. When all the points on the attractor are covered, value of FBI gives  $N(\varepsilon)$  and  $D_{\text{cap}}$  is evaluated from (4). Practically the relation between  $\ln[N(\varepsilon)]$  and  $\ln(1/\varepsilon)$  is linear only over a limited range of  $\varepsilon$ . For larger values of  $\varepsilon$ , finite size of attractor causes  $N(\varepsilon)$  to saturate, while for smaller values of  $\varepsilon$ , finite size of data set causes  $N(\varepsilon)$  to fall sharply. Therefore in practice  $\varepsilon$  is varied over a range and corresponding  $N(\varepsilon)$  is noted each time. From the slope of the linear portion of the plot  $\ln[N(\varepsilon)]$  against  $\ln(1/\varepsilon)$ ,  $D_{\text{cap}}$  is determined.

### 3. Results and discussion

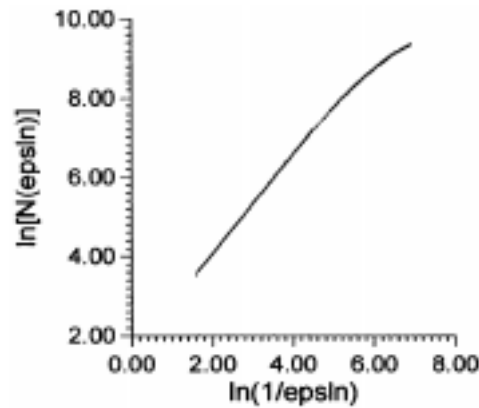
We have applied this algorithm to data set obtained from sampling a sine wave at regular intervals and to the data set obtained from henon map:

**Table 2.** Comparison of results produced by the code.

Embedding dimension	Signal	Results of least square fit of $Y = B * X + A$ in the linear portion of $\ln[N(\epsilon)] - \ln(1/\epsilon)$ curve	Capacity dimension (from code)	Capacity dimension (exact)
3	Sine signal	Equation: $Y = 0.997999 * X + 1.72325$ coeff. of determination, $R$ -squared = 0.997183	0.997999	1.0
3	Henon map	Equation: $Y = 1.25936 * X + 1.55148$ coeff. of determination, $R$ -squared = 0.999463	1.25936	1.26 [7]



**Figure 1.** Dimension calculation for sine signal.



**Figure 2.** Dimension calculation for henon map.

$$X_{n+1} = 1 - aX_n^2 + Y_n, \tag{6a}$$

$$Y_{n+1} = bX_n \tag{6b}$$

with  $a = 1.4$  and  $b = 0.3$ .

In each case a total of 20000 data points were used. Result of computation is shown in table 2. Figures 1 and 2 shows the plot of  $\ln[N(\epsilon)]$  against  $\ln(1/\epsilon)$  for the sine signal and the henon map respectively. Capacity dimension is computed from the slope of the linear portion of the plot determined from a least square fit. It is seen that results produced by the code using this algorithm fairly matches with the established results.

Memory requirement in this algorithm is reduced by noting only the boxes that are filled, using special indexing technique. This also takes the advantage of definition of capacity dimension, which counts only the number of boxes that are filled and does not bother about the number of points in a box. This algorithm can easily be extended to estimate other dimension also using little modification. For example, to determine correlation dimension it needs another array of size  $N(\epsilon)$  to register number of points in corresponding boxes.

The speciality of this algorithm is that it does not depend on the dimension of the state space used to reconstruct the attractor. In practice, it may not be found true. This is because as one increases dimension of the state space from a very low value, the reconstructed attractor gradually unfolds [8]. This causes  $N(\varepsilon)$  to change with  $d$  for a fixed  $\varepsilon$ . When  $d$  is sufficiently large, further unfolding stops and  $N(\varepsilon)$  becomes independent of  $d$ . In fact for correct evaluation of dimension, the attractor should be allowed to get unfolded properly by choosing proper value of  $d$ . To do this practically, one can vary  $d$  over a range starting from a low value and note the value of  $D_{\text{cap}}$  so obtained in each case. At higher side of  $d$ ,  $D_{\text{cap}}$  get fairly stabilized to the exact dimension of the attractor.

#### **4. Summary and conclusion**

To summarize, we have presented in this paper a new algorithm for computing capacity dimension of a dynamical system. The algorithm offers least requirement of memory, flexibility in computation in higher dimensional state space and simplicity in the underlying technique in comparison to the other existing algorithms. The algorithm is primarily meant to boost up the use of capacity dimension, which provides dimension closest to the actual dimension of a system compared to others. However, it can easily be extended to the computation of other dimensions, such as correlation dimension, information dimension etc. with little modifications.

#### **Acknowledgements**

We thankfully acknowledge the constant encouragement from our group members, P S S Murthy, S N Sahasrabudde and M M V Murthy in carrying out this work. Thanks are also due to Dr S K Sikka, Group Director, SSSG for allowing us to carry out this work.

#### **References**

- [1] A Renyi, *Probability Theory* (North-Holland, Amsterdam, 1970)
- [2] A Block, W V Bloh and H J Schellnhuber, *Phys. Rev.* **A42**, 1869 (1990)
- [3] L S Liebovitch and T Toth, *Phys. Lett.* **A147**, 386 (1989)
- [4] A V Aho, J E Hopcroft and J D Ullman, *The design and analysis of computer algorithms* (Addison-Wesley, Reading, MA, 1974)
- [5] X J Hou, R Gilmore, G B Mindjin and H G Solari, *Phys. Lett.* **A151**, 43 (1990)
- [6] T C A Molteno, *Phys. Rev.* **E48**, R3263 (1993)
- [7] A Wolf, J B Swift, H L Swinney and J A Vastano, *Physica* **D16**, 285 (1985)
- [8] J C Roux, R H Simoyi and H L Swinney, *Physica* **D8**, 257 (1983)