

5-28-1991

The Dimension of a Chaotic Attractor

Roslyn Gay Lindquist
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Physics Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Lindquist, Roslyn Gay, "The Dimension of a Chaotic Attractor" (1991). *Dissertations and Theses*. Paper 4182.

<https://doi.org/10.15760/etd.6066>

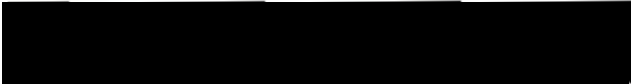
This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

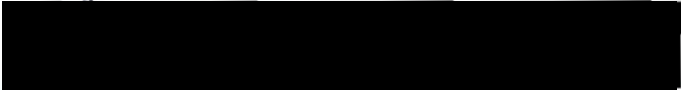
AN ABSTRACT OF THE THESIS OF Roslyn Gay Lindquist
for the Master of Science in Physics presented
May 28, 1991.

Title: The Dimension of a Chaotic Attractor.

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:


Carl Bachhuber, Chair


Eric Bodegom


Jack Semura


David Cox

Tools to explore chaos are as far away as a
personal computer or a pocket calculator. A few
lines of simple equations in BASIC produce fantastic
graphic displays. In the following computer experi-

ment, the dimension of a strange attractor is found by three algorithms; Shaw's, Grassberger-Procaccia's and Guckenheimer's. The programs were tested on the Henon attractor which has a known fractal dimension. Shaw's and Guckenheimer's algorithms were tested with 1000 data points, and Grassberger's with 100 points, a data set easily handled by a PC in one hour or less using BASIC or any other language restricted to 640K RAM. Since dimension estimates are notorious for requiring many data points, the author wanted to find an algorithm to quickly estimate a low-dimensional system (around 2). Although all three programs gave results in the neighborhood of the fractal dimension for the Henon attractor, $D_{\text{fractal}}=1.26$, none appeared to converge to the fractal dimension.

THE DIMENSION OF A CHAOTIC ATTRACTOR

by

ROSLYN GAY LINDQUIST


A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
PHYSICS


Portland State University
1991

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of Roslyn Gay Lindquist presented May 28, 1991.


Carl Bachhuber, Chair


Eric Bodegom


Jack Semura


David Cox

APPROVED:


Mark Gurevitch, Chair, Department of Physics

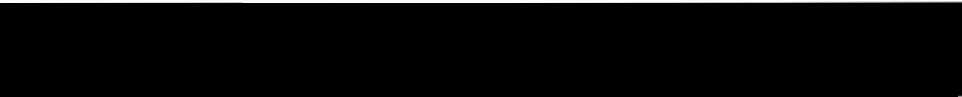

C. William Savery, Interim Vice Provost for Graduate Studies and Research

TABLE OF CONTENTS

	PAGE
LIST OF TABLES.....	III
LIST OF FIGURES.....	IV
CHAPTER	
I INTRODUCTION.....	1
Strange Attractors.....	1
Henon Attractor	
Dimension.....	3
Shaw's Method	
Grassberger-Procaccia Method	
Guckenheimer's Method	
Sources of Error	
II EXPERIMENT.....	10
Dim1.....	10
Dim2.....	11
Dim3.....	12
Henon.....	12
III RESULTS.....	13
Henon Attractor.....	13
Shaw.....	14
Grassberger.....	16
Guckenheimer.....	18
IV DISCUSSION.....	20

REFERENCES..... 22

APPENDICES

A Computer Program "Henon2"..... 23

B Computer Program "Dim1"..... 25

C Computer Program "Dim2"..... 40

D Computer Program "Dim3"..... 54

LIST OF TABLES

TABLE		PAGE
I	Slopes of Lines in Figure 2, Shaw's Method, for Embedding Dimensions 1-5.....	16
II	Slopes of Lines in Figure 3, Grassberger's Method, for Embedding Dimensions 1-5.....	17
III	Slopes of Lines Using X and Y Coordinates, Guckenheimer's Method, for Embedding Dimensions 1-5.....	19

LIST OF FIGURES

FIGURE		PAGE
1.	Henon attractor for $a=1.4$, $b=0.3$, 1000 iterations.....	13
2.	Shaw's method.....	16
3.	Grassberger's method.....	17
4.	Guckenheimer's method.....	19

CHAPTER I

INTRODUCTION

Tools to explore chaos are as far away as a personal computer or a pocket calculator. A few lines of simple equations in BASIC produce fantastic graphic displays. In the following computer experiment, the dimension of a strange attractor is found by three algorithms; Shaw's, Grassberger-Procaccia's and Guckenheimer's. The programs were tested on the Henon attractor which has a known fractal dimension. Shaw's and Guckenheimer's algorithms were tested with 1000 data points, and Grassberger's with 100 points, a data set easily handled by a PC in one hour or less using BASIC or any other language restricted to 640K RAM. Since dimension estimates are notorious for requiring many data points, the author wanted to find an algorithm to quickly estimate a low-dimensional system (around 2).

STRANGE ATTRACTORS

Attractors are points or cycles in phase space that a system is drawn to. If a system with a non-integer dimension has an attractor, it is called a

strange attractor. A strange attractor is chaotic if two points on the attractor separated by a small distance move apart exponentially with time (1).

Attractors exist in phase space. In phase space, each point on the plot describes the state of the total system at one time. An example of an attractor that is not strange is the motion of a simple pendulum (2). In phase space, plot the angle the pendulum makes with the vertical vs the rate at which the angle changes. The point on the graph will circle the origin, spiraling toward the center as friction slows the pendulum. In this case, the origin is the attracting point. If you kick the pendulum, it will eventually return to its stately spiral toward the attracting point.

Henon Attractor

The Henon attractor was first noticed by M. Henon in 1975 (3) while he was looking for a simple mapping with which to study the Lorenz system. The Lorenz attractor is described by three differential equations:

$$dx/dt = a(y - x)$$

$$dy/dt = -xz + bx - y$$

$$dz/dt = xy - cz .$$

Lorenz found this attractor accidentally in 1961

while working on a computer simulation to predict the weather.

Henon wanted to reduce the Lorenz attractor to a Poincare two-dimensional mapping while preserving the properties of the attractor. The mapping consists of a series of foldings and contractions which result in:

$$x_{i+1} = Y_i + 1 - ax_i^2$$

$$Y_{i+1} = bx_i ,$$

now known as the Henon attractor. Henon also showed that these two equations are the most general quadratic mapping that preserve properties of the Lorenz attractor.

DIMENSION

The dimension of an object relates how its volume changes as its linear size changes:

$$\text{BULK} \approx \text{SIZE}^{\text{dimension}} \quad (4).$$

Consider a system with N states, each of radius r . The radius of the whole system is R . Then the dimension, d , is given by:

$$N = (R / r)^d .$$

Taking the log of both sides yields:

$$\log(N) = d \log(R / r),$$

where $\log(N)$ is the stored information and $\log(R/r)$

is the resolution along a single coordinate (5). Let $R=1$ and $r<1$, then the fractal dimension, D_F , is:

$$D_F = \lim_{r \rightarrow 0} (\ln(N) / \ln(1/r)) \quad (6).$$

The fractal dimension is smaller than the degrees of freedom.

D_F is very difficult to calculate when $D_F > 2$. Furthermore, the use of a single time series to extract D_F when $D_F > 2$ has been found to be impractical (1).

The Hausdorff-Besicovich dimension, D_H is another type of dimension used to compare the "size" of systems with the same fractal dimension. The volume of the set may only have three values: zero, infinity, or a finite number. Theiler gives an excellent description of D_H :

For instance, the fractal coastline may have a one-dimensional volume (length) of infinity and a two-dimensional volume (area) of zero, but there is a D between 1 and 2 at which the D volume crosses over from ∞ to 0, and that value of D is the Hausdorff dimension of the coastline (4, p.1060).

The problem with using D_H in experiments is that it requires one to take a minimum over all coverings. The fractal dimension can be used experimentally since it allows a covering of a fixed size grid. The fractal dimension, also called the capacity or the box-counting dimension, is an upper bound on the Hausdorff dimension.

Many ways have been used to find the dimension of an attractor. Hediger found the average local intrinsic dimension of an attractor by counting the number of local orthogonal axis along which points were distributed (7). Some methods divide phase space into boxes or balls and count the number of points in each box or sphere. The three used in this experiment are Shaw's (5), Grassberger-Procaccia's (8), and Guckenheimer's (9) methods.

Shaw's Method

Shaw chose his algorithm for simplicity. It is close to Guckenheimer's and Grassberger's. From a set of time series data, a random point is chosen. The distance from that point to all other points is measured. This is repeated for a number of other randomly chosen points. A mean distance to the nearest neighbor, next nearest neighbor, etcetera, is calculated. $\ln(n)$ vs $\ln(r)$ is plotted where n is the number of data points within a distance r . This process is repeated, grouping the time series data in pairs and plotting $\ln(n)$ vs $\ln(r)$. The data is then grouped in threes, fours, etcetera. This grouping corresponds to increasing the embedding dimension of the attractor. When the embedding dimension is high enough to accommodate the full attractor, the slope of the lines (the dimension) will approach

the dimension of the attractor (5).

Grassberger-Procaccia Method

The Grassberger-Procaccia method, also called the correlation method, is based on the distances between points on the attractor. The correlation integral (8) is written:

$$C(r) = \langle B_X(r) \rangle,$$

where $B_X(r)$ is the pointwise mass function, defined as $B_X(r) = \mu(b_X(r))$. $b_X(r)$ is a ball of radius r , centered at point x . An average is taken since $B_X(r)$ is not uniform over most fractals. The correlation dimension, D_C , is defined:

$$D_C = \lim_{r \rightarrow 0} (\log C(r) / \log(r)) .$$

For a finite data set, $B_X(r)$ can be approximated by:

$$B_{X_j}(r) \approx (1/(N-1)) \sum_{\substack{i=j \\ i \neq j}}^N \theta(r - \|X_i - X_j\|) .$$

θ is the Heavyside step function which is zero for $x < 0$ and one for $x \geq 0$. N is the number of points.

$\|X_i - X_j\|$ is a metric such as the Euclidian metric.

Then

$$\begin{aligned} C(N, r) &= 1/N \sum_{j=1}^N B_{X_j}(r) \\ &= 1/(N(N-1)) \sum_{i \neq j} \theta(r - \|X_i - X_j\|) . \end{aligned}$$

As Theiler put it in words (5),

$$C(N,r) = \frac{\# \text{ of distances less than } r}{\# \text{ of distances altogether}}$$

$C(N,r)$ ranges from $2/N^2$ to one (4).

D_C can be found by taking the slope:

$$D_C = \frac{dC(N,r)/dr}{C(N,r)/r} .$$

The slope is always less than D_F .

Since a finite sample is taken, the sample size is an issue. The minimum number of points has been suggested at:

$$N_{\min} = N_0 A^D ,$$

where A is of the order of ten, but as Theiler points out "experience also indicates the need for more experience" (4,p.1068). The number of points needed to find the dimension of an attractor increases exponentially with increasing dimension (10).

D_I is the information dimension. It represents the information capacity of the attractor. The information dimension lies between D_C and D_H ,

$$D_C \leq D_I \leq D_H .$$

D_F and D_H , taken together, give a good estimate of the information content.

Grassberger found D_C for the Henon attractor to be $1.21 \pm .01$ for 15,000 iterations, and $1.25 \pm .02$ for 20,000 iterations, compared to $D_F=1.26$. Convergence

was apparent for D_C after a few 1000 points (8).

Guckenheimer's Method

Guckenheimer began just as Shaw did, by finding distances to nearest neighbors on the attractor. He sorted the list of distances from large to small and plotted $\log(r)$ vs $\log(V)$. $V=i/N$, where the i th largest distance ($i=1$ to N) has the value r , and N is the total number of points. " i " is the index such that $i=4$ refers to the fourth smallest point. The inverse of the slope is the dimension D_G (9). This technique was developed for use with large data sets that sample all regions of an attractor, and Guckenheimer cautions that his method "must be used with great caution and does not always give reliable results" (9,p 1439-1440).

Sources of Error

Errors occur chiefly because of two properties of attractors and computers. Attractors have infinite structure in a finite space. Computers can only handle a limited number of points.

Infinite structure. Shaw is critical of dimension measurements, pointing out that different parts of an attractor may yield different values because of the limits to which an attractor of infinite structure can be calculated (5).

Edge effects. According to Brandstater, dimension estimates may be low due to edge effects. For a ball r whose center is located near an edge, the increase on the number of points inside is slower than for a ball located far from an edge (10). Theiler writes "the finite sample size leads to poor statistics at small r , and the finite size of the attractor (the edge effect) limits the scaling at large r " (4,p 1069).

Number of points. The number of points required to resolve an attractor down to a certain length scale, increase exponentially as the dimension increases (10).

Non-uniform. Different regions on the attractor exhibit different scaling behavior (10).

CHAPTER II

EXPERIMENT

Three programs were written in True Basic IBM version 1.0 to calculate the dimension of the Henon attractor. A fourth program displays the Henon attractor in phase space. Listings of the programs may be found in the appendices

DIM1

DIM1 uses Shaw's method. It takes 1000 points from the Henon attractor and stores them in an array POINTS. The points chosen were the x coordinates, since any variable of a dynamical system should be able to reconstruct the attractor.

The subroutine MAKE_XDATA1 calculates the distances between a point and its neighbors for all 1000 points using the Euclidian metric. It then calculates the mean distance between a point and its neighbors, storing this data in the array XDATA1.

SORT_XDATA1 sorts the mean distances from smallest to largest.

The PLOT_LINE1 subroutine plots the log of the mean distance vs the log of the number of distances less than that distance.

The slope of the line is found by linear regression. The correlation coefficient of the slope is also found.

Subroutines LINE2 through LINE5 repeat the above process for pairs of points, triplets, fours, and fives.

DIM2

Grassberger and Procaccia's correlation method was used in the program DIM2. Only 100 points from the Henon attractor were used because of the large amount of memory required.

The array POINTS was formed just as in DIM1. Distance between points was calculated and put in XDATA1.

The sum of the distances less than r , $C(N,r)$, is the correlation function. " r " ranges from 0.1 to 2.0. 100 values of r were used.

PLOT_LINE1 draws a graph of $\log(r)$ vs $\log(C(N,r))$.

The slope and its correlation coefficient are calculated, and subroutines LINE2 through LINE5 are

called.

DIM3

DIM3 uses Guckenheimer's method.

POINTS is created, as in the two previous programs. 1000 points are computed.

The SORT_XDATA subroutine sorts the distances in XDATA1 from smallest to largest.

A graph is drawn of $\log(\text{ith distance}/N)$ vs $\log(r)$, where N is the total number of data points, and r is the i th largest distance.

The dimension calculated is the inverse of the slope. The correlation coefficient is found.

LINE2 through LINE5 repeat the process for embedding dimensions 2 through 5.

DIM3 was also run for the y coordinates of the Henon attractor.

HENON

The program HENON plots the Henon attractor,

$$x_1 = 1 - ax^2 + y$$

$$y_1 = bx$$

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

CHAPTER III

RESULTS

HENON ATTRACTOR

The Henon attractor is plotted in figure 1 with the constants $a=1.4$ and $b=0.3$, iterated 1000 times.

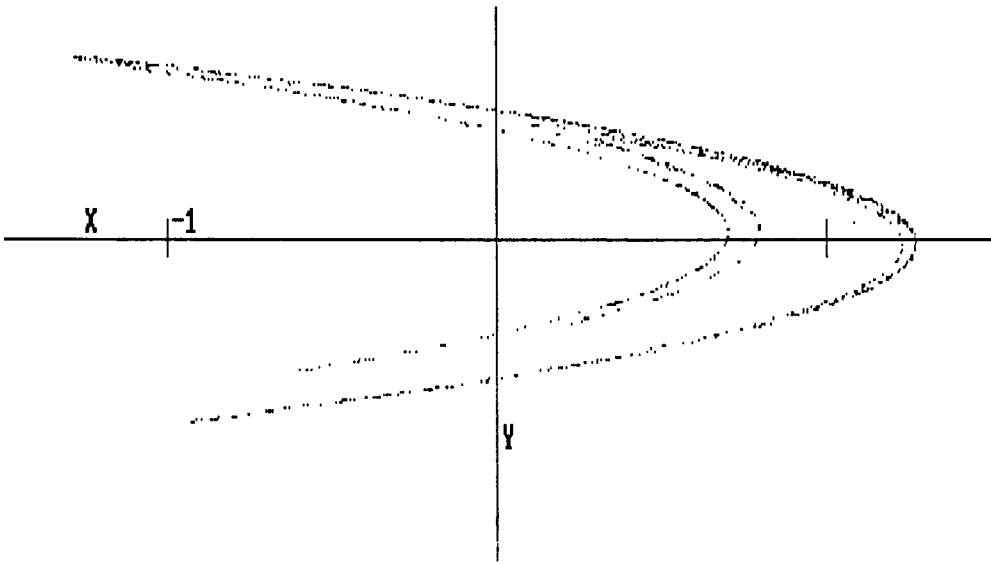


Figure 1. Henon attractor for $a=1.4$,
 $b=0.3$, 1000 iterations.

For these values of a and b , the attractor is chaotic and has a fractal dimension of about 1.26 (1,8).

Initial values of $y=.1$ and $x=.1$ were used, although the picture will appear the same for any starting points chosen.

SHAW

Shaw's algorithm did not converge in 1000 data points to the fractal dimension, 1.26, but gave a value around 0.60. Since Shaw gave no indication how many data points his method takes to converge on a dimension near 2, I can only guess that many more than 1000 are needed.

Figure 2 shows the lines for embedding dimensions 1 through 5. The left line is for 1, the next for 2, etcetera. Table I lists the slopes of the lines.

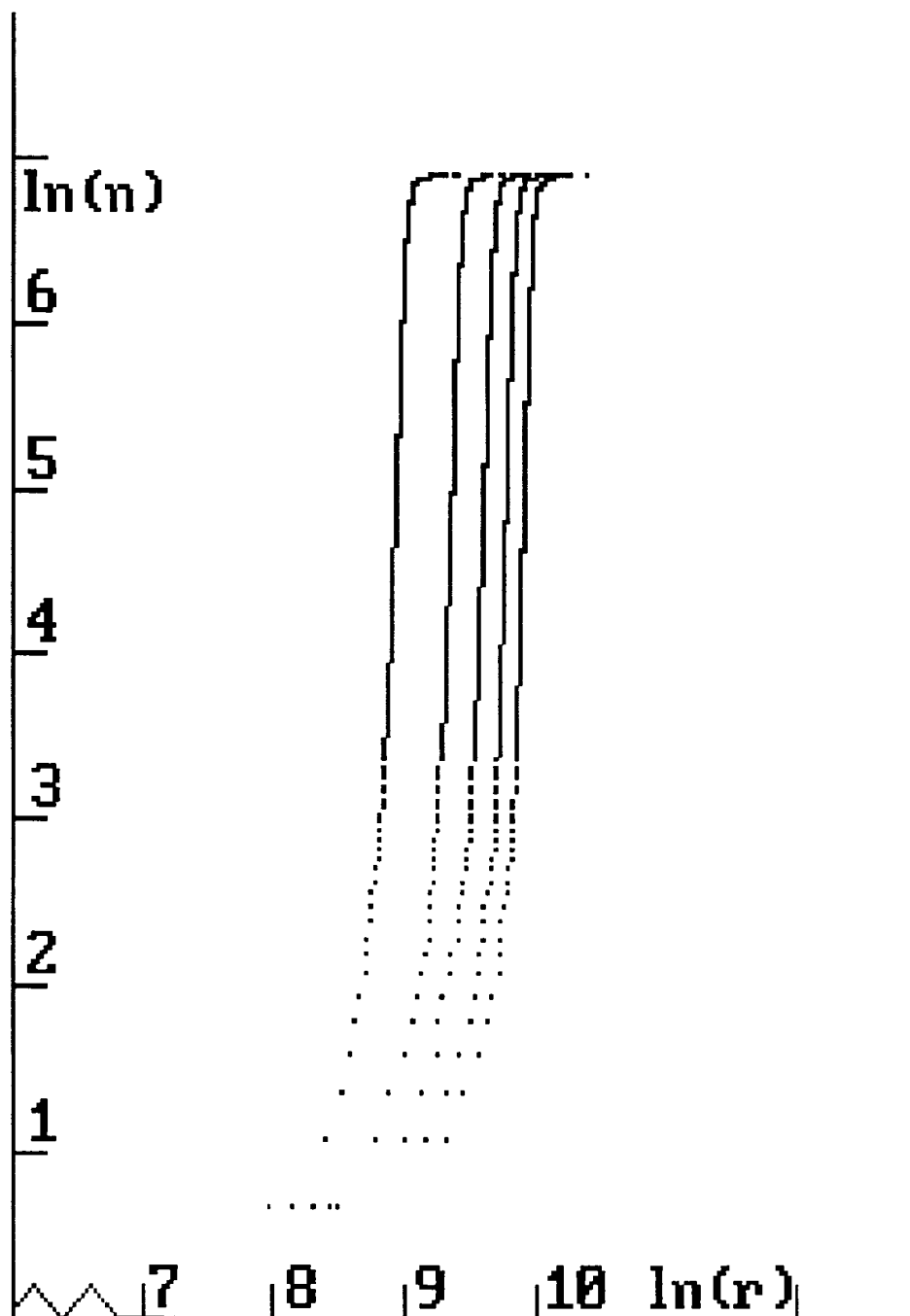


Figure 2. Shaw's method.

TABLE I

SLOPES OF LINES IN FIGURE 2, SHAW'S METHOD,
FOR EMBEDDING DIMENSIONS 1-5³

<u>EMBED. DIM.</u>	<u>SLOPE</u>
1	0.658296
2	0.629123
3	0.613265
4	0.602572
5	0.594524

GRASSBERGER

The Grassberger-Procaccia method overestimated the fractal dimension. The algorithm took the most RAM to run, and so was restricted to 100 data points from the attractor. Grassberger used 20,000 iterations to obtain a value close to the fractal dimension.

Figure 3 shows the lines obtained and Table II lists the slopes. The top line has an embedding dimension of one, the next of 2, and so on.

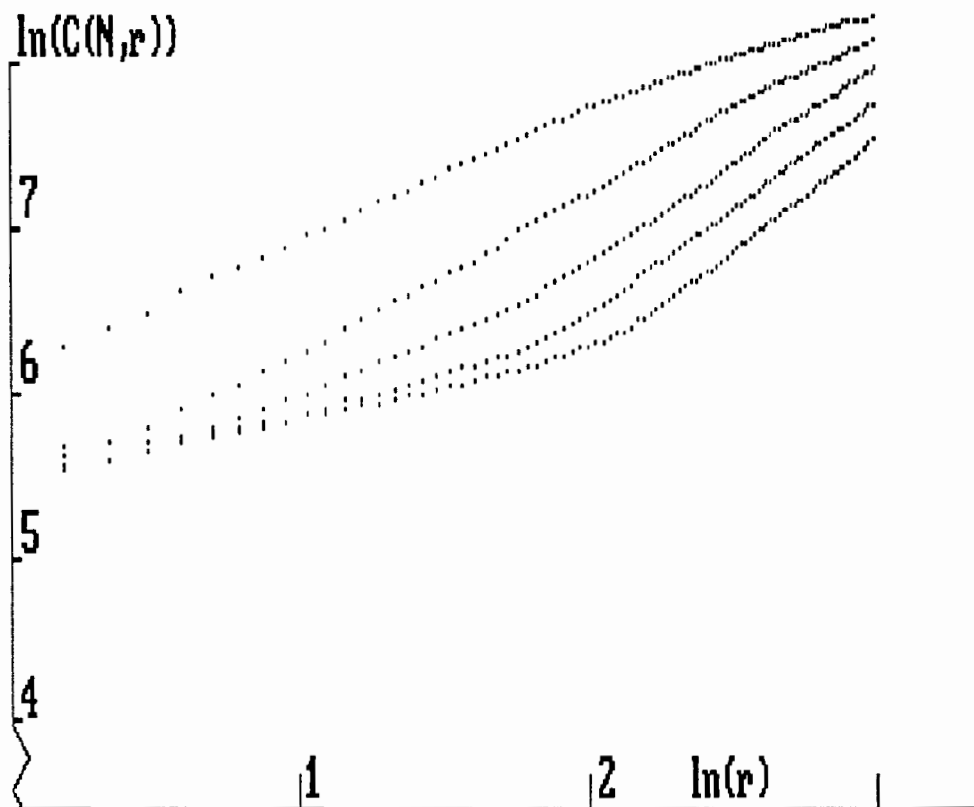


Figure 3. Grassberger's method.

TABLE II

SLOPES OF LINES IN FIGURE 3, GRASSBERGER'S METHOD,
FOR EMBEDDING DIMENSIONS 1-5

<u>EMBED. DIM.</u>	<u>SLOPE</u>
1	3.09066
2	2.97917
3	2.86125
4	2.75760
5	2.66498

GUCKENHEIMER

For 1000 data points, Guckenheimer's method gave a value in the neighborhood of the fractal dimension, as did the other two methods.

The slopes found using the x coordinate of the attractor were greater than the fractal dimension. Figure 4 graphs the lines found using the x coordinates. The left line is for an embedding dimension fo 1, the next for 2, and so on. Repeating the same program with the y coordinates, the slopes were less than the fractal dimension. The slopes using the y coordinates appeared to be converging more quickly than the x coordinates as shown in Table III.

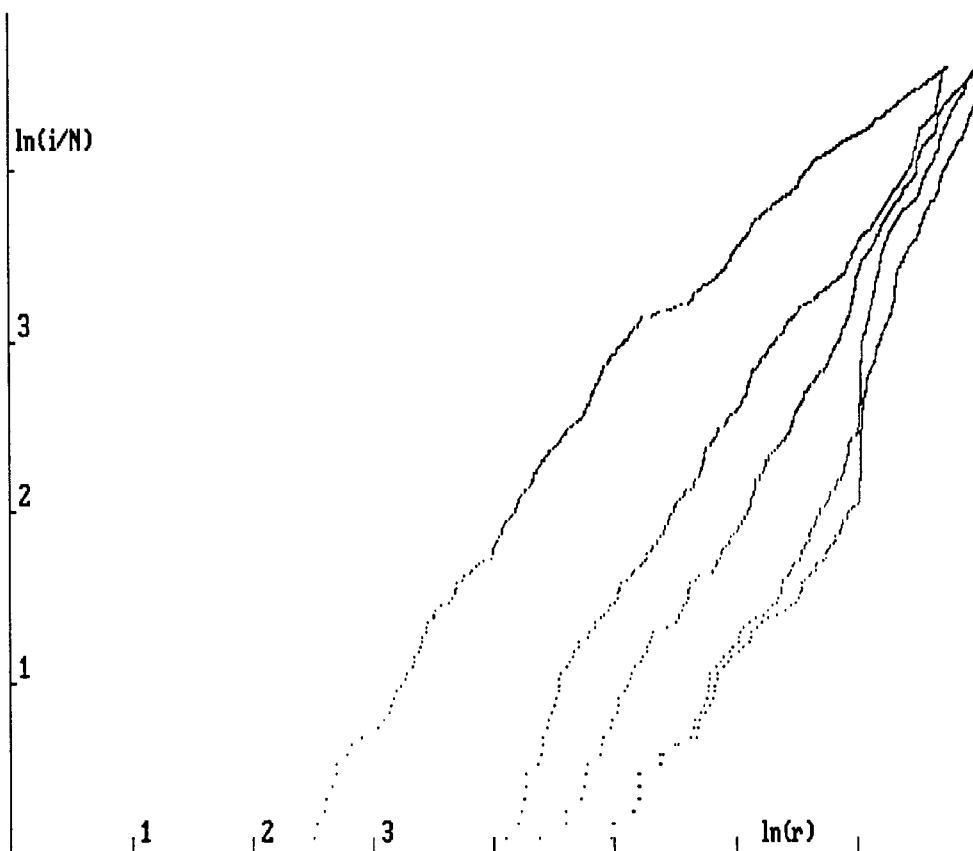


Figure 4. Guckenheimer's method.

TABLE III

SLOPES OF LINES USING X AND Y COORDINATES,
 GUCKENHEIMER'S METHOD, FOR EMBEDDING
 DIMENSIONS 1-5

<u>EMBED. DIM.</u>	<u>X SLOPE</u>	<u>Y SLOPE</u>
1	1.68290	0.941998
2	1.87335	0.952682
3	1.93501	0.953968
4	1.99319	0.954323
5	2.03469	0.954594

CHAPTER IV

DISCUSSION

Although all three methods gave results in the neighborhood of the fractal dimension, no one algorithm stood out as the best for a small number of data points.

Shaw's method took more run-time than Guckenheimer's method because of the extra step of finding the mean distance to neighbors, but this step gave a much lower estimate of the dimension.

Grassberger's algorithm required large arrays and more computing time than the other two methods. Fewer data points from the attractor could be used. The slopes at higher embedding dimension did not appear to be converging as did the slopes in the other two methods.

Guckenheimer's method took the least time to run. The x coordinates appeared to be converging at a slope around 2, and the y coordinates around 1. Since this algorithm was designed for very large data sets, the difference in slopes between the

x and y data sets may not be significant.

None of the three methods came close enough to the fractal dimension to be useful in the determination of dimension for a low dimensional attractor using few data points.

REFERENCES

- (1) Grebogi, Celso, Ott, Edward, Yorke, James A., "Chaos, strange attractors, and fractal basin boundaries in nonlinear dynamics," Science **238**, 632 (1987).
- (2) Dewdney, A.K., "Probing the strange attractors of chaos," Sci. Amer. **257**, 108 (1987).
- (3) Dold, A., Eckmann, B., Turbulence and Navier Stokes Equation, Orsay 1975 (Springer Verlag, Berlin, 1976).
- (4) Theiler, James, "Estimating Fractal Dimension," J. Opt. Soc. Am. A **7**, 1055 (1990).
- (5) Shaw, Robert, The Dripping Faucet as a Model Chaotic System (Aerial Press, Santa Cruz, 1984).
- (6) Barnsley, Michael, Fractals Everywhere (Academic Press, San Diego, 1988).
- (7) Hediger, T., Passamante, A., Farrell, Mary Eileen, "Characterizing attractors using local intrinsic dimensions calculated by singular-value decomposition and information-theoretic criteria," Phys. Rev. A **41**, 5325 (1990).
- (8) Grassberger, Peter, Procaccia, Itmar, "Characterization of strange attractors," Phys. Rev. Lett. **50**, 346 (1983).
- (9) Guckenheimer, John, Buzna, George, "Dimension measurements for geostrophic turbulence," Phys. Rev. Lett. **51**, 1438 (1983).
- (10) Brandstater, A., Swinney, Harry L., "Strange attractors in weakly turbulent Couette-Taylor flow," Phys. Rev. A **35**, 2207 (1987).

APPENDIX A

COMPUTER PROGRAM "HENON2"

```

REM henon2
SET MODE "egahires"
SET WINDOW -1.5,1.5,-1,1
LET y=.1                !initial values
LET x=.1
LET a=1.4                !constants
LET b=.3

! *****
! *                      SUB AXIS                      *
! *      Draws x and y axis.                          *
! *****

SUB axis
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
END SUB

! *****
! *                      SUB TICKS                      *
! *      Draws ticks on x and y axis.                  *
! *****

SUB ticks(x,y)          ! axis with ticks
x and y units apart
  ASK SCREEN u1,u2,v1,v2
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
  LET xu=640
  LET p1=abs(v2-v1)*200    !pixels vertical
  LET p2=abs(u2-u1)*xu    !pixels horizontal
  LET r=min(p1/50,p2/50)  !small fraction
  LET d1=r/p1*abs(y2-y1)  !for x
  LET d2=r/p2*abs(x2-x1)  !for y
  CALL mark(x1,x,1,d1)
  CALL mark(x2,x,1,d1)
  CALL mark(y1,y,2,d2)
  CALL mark(y2,y,2,d2)
END SUB

```



```

! *****
! *                               SUB MARK                               *
! *   Used by SUB TICKS to draw a single tick.   *
! *****

```

```

SUB mark(u2,us,c,d)                !does 1 tick
  IF u2=0 then EXIT SUB
  FOR u=0 to u2 step sgn(u2)*us
    IF c=1 then
      PLOT u,-d;u,d
    ELSE
      PLOT -d,u;d,u
    END IF
  NEXT u
END SUB

```

```

! *****
! *                               MAIN PROGRAM                               *
! *****

```

```

SET COLOR "white"
CALL axis
CALL ticks (1,1)
FOR i=1 to 1000
  LET xx=1+y-a*x*x
  LET yy=b*x
  LET x=xx
  LET y=yy
  PLOT (x),(y)
NEXT i
END

```

APPENDIX B

COMPUTER PROGRAM "DIM1"

```
! *****
! *                               DIM1                               *
! *   Finds the dimension of the Henon attractor                       *
! *   using Shaw's method.                                             *
! *   Slope of lines plotted approaches the                             *
! *   dimension of the attractor.                                       *
! *****
```

```
! *****
! *   Initialize variables and constants. Declare                       *
! *   arrays.                                                           *
! *****
```

```
OPEN #1: printer
SET MODE "egahires"
SET WINDOW 0,20,0,10
DIM points(1000,1)
DIM xdata1(999,3)                !one less than #
of data pts
DIM xdata2(998,3)                ! 2 less xdata1
DIM xdata3(997,3)
DIM xdata4(996,3)
DIM xdata5(995,3)
LET y=.1
LET x=.1                        !initial value
LET a=1.4                        !constant
LET b=.3                         !constant
LET total_data_pts=1000         !# of data pts
```

```
! *****
! *                               SUB AXIS                               *
! *   Draws x and y axis.                                             *
! *****
```

```
SUB axis
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
END SUB
```

```
! *****
! *                SUB TICKS                *
! *      Draws ticks on x and y axis.      *
! *****
```

```
SUB ticks(x,y)                ! axis with ticks
x and y units apart
  ASK SCREEN u1,u2,v1,v2
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
  LET xu=640
  LET p1=abs(v2-v1)*200      !pixels vertical
  LET p2=abs(u2-u1)*xu      !pixels horizontal
  LET r=min(p1/50,p2/50)    !small fraction
  LET d1=r/p1*abs(y2-y1)    !for x
  LET d2=r/p2*abs(x2-x1)    !for y
  CALL mark(x1,x,1,d1)
  CALL mark(x2,x,1,d1)
  CALL mark(y1,y,2,d2)
  CALL mark(y2,y,2,d2)
END SUB
```

```
! *****
! *                SUB MARK                *
! *      Used by SUB TICKS to draw a single tick.  *
! *****
```

```
SUB mark(u2,us,c,d)          !does 1 tick
  IF u2=0 then EXIT SUB
  FOR u=0 to u2 step sgn(u2)*us
    IF c=1 then
      PLOT u,-d;u,d
    ELSE
      PLOT -d,u;d,u
    END IF
  NEXT u
END SUB
```

```
! *****
! *                SUB FUNCTION            *
! *      Equations for Henon attractor.    *
! *****
```

```
SUB function (x)
  LET xx=1-a*x*x+y
  LET yy=b*x
  LET x=xx
  LET y=yy
END SUB
```

```

! *****
! *          SUB MAKE_POINTS          *
! *      Reads x & y values of strange attractor *
! *      equations into an array POINTS *
! *****

```

```

SUB make_points (x)
  FOR i=1 to total_data_pts      !# of data pts
    CALL function (x)
    LET points (i,1)=x
  NEXT i
END SUB

```

```

! *****
! *          SUB INITIALIZE_ARRAY      *
! *      Sets all elements in the arrays *
! *      to zero.                      *
! *****

```

```

SUB initialize_array
  MAT points=0
  MAT xdata1=0
  MAT xdata2=0
  MAT xdata3=0
  MAT xdata4=0
  MAT xdata5=0
END SUB

```

```

! *****
! *          SUB MAKE_xdata1          *
! *      Forms the array xdata1. Finds the *
! *      distance between each point and its *
! *      neighbor. The index of the array stands *
! *      for the neighbor. Distances are stored *
! *      in the first column. Number of times *
! *      the distance to each neighbor is *
! *      measured is stored in the second *
! *      column. The mean distance to each *
! *      neighbor, calculated by dividing *
! *      column 1 by column 2, is placed in the *
! *      third column.                *
! *****

```

```

SUB make_xdata1
  LET pointnum=0
  FOR m=1 to total_data_pts
    LET start1x=points(m,1)
    FOR j=m+1 to total_data_pts
      LET pointnum=pointnum+1
      LET dif1=(start1x-points(j,1))*(start1x-
points(j,1))
      LET difsq=sqr(dif1)

```

```

                LET xdata1(pointnum,1)=difsq +
xdata1(pointnum,1)
                LET
xdata1(pointnum,2)=xdata1(pointnum,2)+1
                NEXT j
                LET pointnum=0
            NEXT m
            FOR i=1 to total_data_pts-1
                IF xdata1(i,2)<>0 then
                    LET xdata1(i,3)=xdata1(i,1)/xdata1(i,2)
!mean
                END IF
            NEXT i
END SUB

```

```

! *****
! *          SUB sort_xdata1          *
! *  Sorts the array xdata1 smallest  *
! *  to largest.                      *
! *****

```

```

SUB sort_xdata1
    FOR first = 1 to total_data_pts-1
        LET smallest=first
        FOR current=first to total_data_pts-1
            IF xdata1(current,3)<xdata1(smallest,3)
then
                LET smallest =current
            END IF
        NEXT current
        LET temp=xdata1(smallest,3)
        LET xdata1(smallest,3)=xdata1(first,3)
        LET xdata1(first,3)=temp
    NEXT first
END SUB

```

```

! *****
! *          SUB PLOT_XLINE1          *
! *  Plots the log of the distance    *
! *  vs the log of the number of points *
! *  less than that distance. Also    *
! *  calculates sums to be used in finding *
! *  the slope in the subroutine      *
! *  FIND_SLOPE.                      *
! *****

```

```

SUB plot_xline1
    LET old_x=0
    LET old_y=0
    LET old_xy=0
    LET old_xx=0
    LET old_yy=0

```

```

FOR i=1 to total_data_pts-1
  IF xdata1(i,3)<>0 then
    LET xcoord=log(xdata1(i,3)*10000) !ln(r)
    LET ycoord=log(i) ! ln(n)
    PLOT xcoord,ycoord
    LET sumx=xcoord+old_x !sum of x coord
    LET old_x=sumx
    LET sumy=ycoord+old_y !sum of y coord
    LET old_y=sumy
    LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
    LET old_xy=sumxy
    LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
    LET old_xx=sumxx
    LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
    LET old_yy=sumyy
  END IF
NEXT i
END SUB

```

```

! *****
! *                               *
! *           SUB XLINE1          *
! *   Calls the routines to calculate and *
! *   plot the first line and return the *
! *   slope.                       *
! *****

```

```

SUB xline1
  CALL make_xdata1
  CALL sort_xdata1
  CALL plot_xline1
  CALL find_xslope
END SUB

```

```

! *****
! *                               *
! *           SUB MAKE_xdata2     *
! *   Forms the array xdata2. Finds the *
! *   distance between pairs of points. *
! *   The index of the array stands *
! *   for the neighbor. Distances are stored *
! *   in the first column. Number of times *
! *   the distance to each neighbor is *
! *   measured is stored in the second *
! *   column. The mean distance to each *
! *   neighbor, calculated by dividing *
! *   column 1 by column 2, is placed in the *
! *   third column.                *
! *****

```

```

SUB make_xdata2
  LET pointnum=0
  FOR m=1 to total_data_pts-4
    LET start1x=points(m,1)
    LET start2x=points(m+1,1)
    FOR j=m+1 to total_data_pts-2
      LET pointnum=pointnum+1
      LET dif1=(start1x-points(j,1))*(start1x-
points(j,1))
      LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
      LET difsq=sqr(dif1+dif2)
      LET xdata2(pointnum,1)=difsq +
xdata2(pointnum,1)
      LET
xdata2(pointnum,2)=xdata2(pointnum,2)+1
    NEXT j
    LET pointnum=0
  NEXT m
  FOR i=1 to total_data_pts-2
    IF xdata2(i,2)<>0 then
      LET xdata2(i,3)=xdata2(i,1)/xdata2(i,2)
!mean
      !print #1:xdata2(i,1),xdata2(i,3)
    END IF
  NEXT i
END SUB

! *****
! *          SUB sort_xdata2          *
! *  Sorts the array xdata2 smallest  *
! *  to largest.                      *
! *****

SUB sort_xdata2
  FOR first = 1 to total_data_pts-2
    LET smallest=first
    FOR current=first to total_data_pts-2
      IF xdata2(current,3)<xdata2(smallest,3)
then
          LET smallest =current
        END IF
    NEXT current
    LET temp=xdata2(smallest,3)
    LET xdata2(smallest,3)=xdata2(first,3)
    LET xdata2(first,3)=temp
  NEXT first
END SUB

```

```

! *****
! *          SUB PLOT_XLINE2          *
! *    Plots the log of the distance  *
! *    vs the log of the number of points *
! *    less than that distance. Also  *
! *    calculates sums to be used in finding *
! *    the slope in the subroutine    *
! *    FIND_SLOPE.                   *
! *****

```

```

SUB plot_xline2
  LET old_x=0
  LET old_y=0
  LET old_xy=0
  LET old_xx=0
  LET old_yy=0
  FOR i=1 to total_data_pts-2
    IF xdata2(i,3)<>0 then
      LET xcoord=log(xdata2(i,3)*10000)      !
ln(r)
      LET ycoord=log(i)          ! ln(n)
      PLOT xcoord,ycoord
      LET sumx=xcoord+old_x      !sum of x coord
      LET old_x=sumx
      LET sumy=ycoord+old_y      !sum of y coord
      LET old_y=sumy
      LET sumxy=xcoord*ycoord+old_xy      !sum of
x*y coord
      LET old_xy=sumxy
      LET sumxx=xcoord*xcoord+old_xx      !sum of
x*x coord
      LET old_xx=sumxx
      LET sumyy=ycoord*ycoord+old_yy      !sum of
y*y coord
      LET old_yy=sumyy
    END IF
  NEXT i
END SUB

```

```

! *****
! *          SUB XLINE2          *
! *    Calls the routines to calculate and *
! *    plot the second line and return the *
! *    slope.                       *
! *****

```

```

SUB xline2
  CALL make_xdata2
  CALL sort_xdata2
  CALL plot_xline2
  CALL find_xslope
END SUB

```



```

! *****
! *          SUB MAKE_xdata3          *
! *    Forms the array xdata3. Finds the *
! *    distance between three points.   *
! *    The index of the array stands   *
! *    for the neighbor. Distances are stored *
! *    in the first column. Number of times *
! *    the distance to each neighbor is *
! *    measured is stored in the second *
! *    column. The mean distance to each *
! *    neighbor, calculated by dividing *
! *    column 1 by column 2, is placed in the *
! *    third column.                   *
! *****

SUB make_xdata3
  LET pointnum=0
  FOR m=1 to total_data_pts-6
    LET start1x=points(m,1)
    LET start2x=points(m+1,1)
    LET start3x=points(m+2,1)
    FOR j=m+1 to total_data_pts-3
      LET pointnum=pointnum+1
      LET dif1=(start1x-points(j,1))*(start1x-
points(j,1))
      LET dif2=(start2x - points(j+1,1)) *
(start2x - points(j+1,1))
      LET dif3=(start3x-points(j+2,1)) *
(start3x - points(j+2,1))
      LET difsq=sqr(dif1+dif2+dif3)
      LET xdata3(pointnum,1)=difsq +
xdata3(pointnum,1)
      LET
xdata3(pointnum,2)=xdata3(pointnum,2)+1
    NEXT j
    LET pointnum=0
  NEXT m
  FOR i=1 to total_data_pts-3
    IF xdata3(i,2)<>0 then
      LET xdata3(i,3)=xdata3(i,1)/xdata3(i,2)
!mean
      !print #1:xdata3(i,1),xdata3(i,3)
    END IF
  NEXT i
END SUB

```

```

! *****
! *          SUB sort_xdata3          *
! *    Sorts the array xdata3 smallest *
! *    to largest.                   *
! *****

```

```

SUB sort_xdata3
  FOR first = 1 to total_data_pts-3
    LET smallest=first
    FOR current=first to total_data_pts-3
      IF xdata3(current,3)<xdata3(smallest,3)
then
          LET smallest =current
          END IF
    NEXT current
    LET temp=xdata3(smallest,3)
    LET xdata3(smallest,3)=xdata3(first,3)
    LET xdata3(first,3)=temp
  NEXT first
END SUB

! *****
! *          SUB PLOT_xline3          *
! *    Plots the log of the distance   *
! *    vs the log of the number of points *
! *    less than that distance. Also   *
! *    calculates sums to be used in finding *
! *    the slope in the subroutine     *
! *    FIND_SLOPE.                    *
! *****

SUB plot_xline3
  LET old_x=0
  LET old_y=0
  LET old_xy=0
  LET old_xx=0
  LET old_yy=0
  FOR i=1 to total_data_pts-3
    IF xdata3(i,3)<>0 then
      LET xcoord=log(xdata3(i,3)*10000)      !
ln(r)          LET ycoord=log(i)          ! ln(n)
                PLOT xcoord,ycoord
                LET sumx=xcoord+old_x      !sum of x coord
                LET old_x=sumx
                LET sumy=ycoord+old_y      !sum of y coord
                LET old_y=sumy
                LET sumxy=xcoord*ycoord+old_xy      !sum of
x*y coord
                LET old_xy=sumxy
                LET sumxx=xcoord*xcoord+old_xx      !sum of
x*x coord
                LET old_xx=sumxx
                LET sumyy=ycoord*ycoord+old_yy      !sum of
y*y coord
                LET old_yy=sumyy
  END IF

```

```

    NEXT i
END SUB

```

```

! *****
! *                               *
! *           SUB xline3          *
! *   Calls the routines to calculate and *
! *   plot the second line and return the *
! *   slope.                      *
! *                               *
! *****

```

```

SUB xline3
  CALL make_xdata3
  CALL sort_xdata3
  CALL plot_xline3
  CALL find_xslope
END SUB

```

```

! *****
! *                               *
! *           SUB MAKE_xdata4     *
! *   Forms the array xdata4. Finds the *
! *   distance between four points.    *
! *   The index of the array stands   *
! *   for the neighbor. Distances are stored *
! *   in the first column. Number of times *
! *   the distance to each neighbor is *
! *   measured is stored in the second *
! *   column. The mean distance to each *
! *   neighbor, calculated by dividing *
! *   column 1 by column 2, is placed in the *
! *   third column.                 *
! *                               *
! *****

```

```

SUB make_xdata4
  LET pointnum=0
  FOR m=1 to total_data_pts-8
    LET start1x=points(m,1)
    LET start2x=points(m+1,1)
    LET start3x=points(m+2,1)
    LET start4x=points(m+3,1)
    FOR j=m+1 to total_data_pts-4
      LET pointnum=pointnum+1
      LET dif1=(start1x-points(j,1))*(start1x-
points(j,1))
      LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
      LET dif3=(start3x-
points(j+2,1))*(start3x-points(j+2,1))
      LET dif4=(start4x-
points(j+3,1))*(start4x-points(j+3,1))
      LET difsq=sqr(dif1+dif2+dif3+dif4)
      LET xdata4(pointnum,1)=difsq +
xdata4(pointnum,1)
    
```

```

                LET
xdata4(pointnum,2)=xdata4(pointnum,2)+1
                NEXT j
                LET pointnum=0
            NEXT m
            FOR i=1 to total_data_pts-4
                IF xdata4(i,2)<>0 then
                    LET xdata4(i,3)=xdata4(i,1)/xdata4(i,2)
!mean
                    !print #1:xdata4(i,1),xdata4(i,3)
                END IF
            NEXT i
        END SUB

```

```

! *****
! *          SUB sort_xdata4          *
! *  Sorts the array xdata4 smallest *
! *  to largest.                      *
! *****

```

```

SUB sort_xdata4
    FOR first = 1 to total_data_pts-4
        LET smallest=first
        FOR current=first to total_data_pts-4
            IF xdata4(current,3)<xdata4(smallest,3)
then
                LET smallest =current
            END IF
        NEXT current
        LET temp=xdata4(smallest,3)
        LET xdata4(smallest,3)=xdata4(first,3)
        LET xdata4(first,3)=temp
    NEXT first
END SUB

```

```

! *****
! *          SUB PLOT_xline4          *
! *  Plots the log of the distance   *
! *  vs the log of the number of points *
! *  less than that distance. Also   *
! *  calculates sums to be used in finding *
! *  the slope in the subroutine     *
! *  FIND_SLOPE.                     *
! *****

```

```

SUB plot_xline4
    LET old_x=0
    LET old_y=0
    LET old_xy=0
    LET old_xx=0
    LET old_yy=0
    FOR i=1 to total_data_pts-4

```

```

      IF xdata4(i,3)<>0 then
      LET xcoord=log(xdata4(i,3)*10000)      !
ln(r)      LET ycoord=log(i)          ! ln(n)
          PLOT xcoord,ycoord
          LET sumx=xcoord+old_x  !sum of x coord
          LET old_x=sumx
          LET sumy=ycoord+old_y  !sum of y coord
          LET old_y=sumy
          LET sumxy=xcoord*ycoord+old_xy  !sum of
x*y coord
          LET old_xy=sumxy
          LET sumxx=xcoord*xcoord+old_xx  !sum of
x*x coord
          LET old_xx=sumxx
          LET sumyy=ycoord*ycoord+old_yy  !sum of
y*y coord
          LET old_yy=sumyy
      END IF
    NEXT i
  END SUB

```

```

! *****
! *              SUB xline4              *
! *    Calls the routines to calculate and *
! *    plot the second line and return the *
! *    slope.                             *
! *****

```

```

SUB xline4
  CALL make_xdata4
  CALL sort_xdata4
  CALL plot_xline4
  CALL find_xslope
END SUB

```

```

! *****
! *              SUB MAKE_xdata5        *
! *    Forms the array xdata5. Finds the *
! *    distance between five points.     *
! *    The index of the array stands    *
! *    for the neighbor. Distances are stored *
! *    in the first column. Number of times *
! *    the distance to each neighbor is *
! *    measured is stored in the second *
! *    column. The mean distance to each *
! *    neighbor, calculated by dividing *
! *    column 1 by column 2, is placed in the *
! *    third column.                    *
! *****

```

```

SUB make_xdata5

```

```

LET pointnum=0
FOR m=1 to total_data_pts-10
  LET start1x=points(m,1)
  LET start2x=points(m+1,1)
  LET start3x=points(m+2,1)
  LET start4x=points(m+3,1)
  LET start5x=points(m+4,1)
  FOR j=m+1 to total_data_pts-5
    LET pointnum=pointnum+1
    LET dif1=(start1x-points(j,1))*(start1x-
points(j,1))
    LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
    LET dif3=(start3x-
points(j+2,1))*(start3x-points(j+2,1))
    LET dif4=(start4x-
points(j+3,1))*(start4x-points(j+3,1))
    LET dif5=(start5x-
points(j+4,1))*(start5x-points(j+4,1))
    LET difsq=sqr(dif1+dif2+dif3+dif4+dif5)
    LET xdata5(pointnum,1)=difsq +
xdata5(pointnum,1)
    LET
xdata5(pointnum,2)=xdata5(pointnum,2)+1
  NEXT j
  LET pointnum=0
NEXT m
FOR i=1 to total_data_pts-5
  IF xdata5(i,2)<>0 then
    LET xdata5(i,3)=xdata5(i,1)/xdata5(i,2)
!mean
    !print #1:xdata5(i,1),xdata5(i,3)
  END IF
NEXT i
END SUB

```

```

! *****
! *          SUB sort_xdata5          *
! *  Sorts the array xdata5 smallest  *
! *  to largest.                      *
! *****

```

```

SUB sort_xdata5
  FOR first = 1 to total_data_pts-5
    LET smallest=first
    FOR current=first to total_data_pts-5
      IF xdata5(current,3)<xdata5(smallest,3)
then
        LET smallest =current
      END IF
    NEXT current
    LET temp=xdata5(smallest,3)

```

```

        LET xdata5(smallest,3)=xdata5(first,3)
        LET xdata5(first,3)=temp
    NEXT first
END SUB

! *****
! *          SUB PLOT_xline5          *
! *    Plots the log of the distance  *
! *    vs the log of the number of points *
! *    less than that distance. Also  *
! *    calculates sums to be used in finding *
! *    the slope in the subroutine    *
! *    FIND_SLOPE.                   *
! *****

SUB plot_xline5
    LET old_x=0
    LET old_y=0
    LET old_xy=0
    LET old_xx=0
    LET old_yy=0
    FOR i=1 to total_data_pts-5
        IF xdata5(i,3)<>0 then
            LET xcoord=log(xdata5(i,3)*10000)      !
ln(r)          LET ycoord=log(i)          ! ln(n)
                PLOT xcoord,ycoord
                LET sumx=xcoord+old_x  !sum of x coord
                LET old_x=sumx
                LET sumy=ycoord+old_y  !sum of y coord
                LET old_y=sumy
                LET sumxy=xcoord*ycoord+old_xy  !sum of
x*y coord
                LET old_xy=sumxy
                LET sumxx=xcoord*xcoord+old_xx  !sum of
x*x coord
                LET old_xx=sumxx
                LET sumyy=ycoord*ycoord+old_yy  !sum of
y*y coord
                LET old_yy=sumyy
            END IF
        NEXT i
    END SUB

! *****
! *          SUB xline5          *
! *    Calls the routines to calculate and *
! *    plot the second line and return the *
! *    slope.                   *
! *****

SUB xline5

```

```

CALL make_xdata5
CALL sort_xdata5
CALL plot_xline5
CALL find_xslope
END SUB

```

```

! *****
! *                SUB FIND_XSLOPE                *
! *    Calculates the slope of a line using        *
! *    linear regression.                          *
! *****

```

```

SUB find_xslope
LET pts=total_data_pts-1
LET sxy=sumxy-((sumx*sumy)/((2*pts)-1))
LET sxx=sumxx-((sumx*sumx)/((2*pts)-1))
LET syy=sumyy-((sumy*sumy)/((2*pts)-1))
LET slope=sxy/sxx
LET r=sxy/(sqr(sxx*syy))
PRINT slope
END SUB

```

```

! *****
! *                MAIN PROGRAM                *
! *    Calls subroutines to create each line.    *
! *****

```

```

SET COLOR "white"
CALL axis
CALL ticks(1,1)
CALL initialize_array
CALL make_points(x)
CALL xline1
SET COLOR "yellow"
CALL xline2
SET COLOR "blue"
CALL xline3
SET COLOR "red"
CALL xline4
SET COLOR "green"
CALL xline5
SOUND 300,.5
END

```


APPENDIX C

COMPUTER PROGRAM "DIM2"

```
! *****
! *                               DIM2                               *
! *   Finds the dimension of the Henon attractor                   *
! *   using the correlation method.                                  *
! *   Slope of lines plotted approaches the                        *
! *   dimension of the attractor.                                   *
! *****

! *****
! *   Initialize variables and constants. Declare                  *
! *   arrays.                                                       *
! *****

OPEN #1: printer
SET MODE "egahires"
SET WINDOW 0,5,0,10
DIM xdata1(4950,1)
DIM points(101,1)
DIM xcor1(100,2)
DIM xcor2(100,2)
DIM xdata2(4900,1)
DIM xdata3(4850,1)
DIM xcor3(100,2)
DIM xcor4(100,2)
DIM xdata4(4800,1)
DIM xdata5(4750,1)
DIM xcor5(100,2)
LET y=.4
LET x=.4                               !initial value
LET a=1.4                               !constant
LET b=.3                               !constant
LET total_data_pts=100                 !# of data pts

! *****
! *                               SUB AXIS                               *
! *   Draws x and y axis.                                           *
! *****
```

```

SUB axis
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
END SUB

```

```

! *****
! *                SUB TICKS                *
! *      Draws ticks on x and y axis.      *
! *****

```

```

SUB ticks(x,y)                ! axis with ticks
x and y units apart
  ASK SCREEN u1,u2,v1,v2
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
  LET xu=640
  LET p1=abs(v2-v1)*200      !pixels vertical
  LET p2=abs(u2-u1)*xu      !pixels horizontal
  LET r=min(p1/50,p2/50)    !small fraction
  LET d1=r/p1*abs(y2-y1)    !for x
  LET d2=r/p2*abs(x2-x1)    !for y
  CALL mark(x1,x,1,d1)
  CALL mark(x2,x,1,d1)
  CALL mark(y1,y,2,d2)
  CALL mark(y2,y,2,d2)
END SUB

```

```

! *****
! *                SUB MARK                *
! *      Used by SUB TICKS to draw a single tick.  *
! *****

```

```

SUB mark(u2,us,c,d)          !does 1 tick
  IF u2=0 then EXIT SUB
  FOR u=0 to u2 step sgn(u2)*us
    IF c=1 then
      PLOT u,-d;u,d
    ELSE
      PLOT -d,u;d,u
    END IF
  NEXT u
END SUB

```

```

! *****
! *                SUB FUNCTION            *
! *      Equations for Henon attractor.      *
! *****

```

```

SUB function (x)
  LET xx=1-a*x*x+y

```

```

    LET yy=b*x
    LET x=xx
    LET y=yy
END SUB

```

```

! *****
! *                SUB MAKE_POINTS                *
! *    Reads x values of strange attractor        *
! *    equations into an array POINTS            *
! *****

```

```

SUB make_points (x)
    FOR i=1 to total_data_pts    !# of data pts
        CALL function (x)
        LET points (i,1)=x
    NEXT i
END SUB

```

```

! *****
! *                SUB INITIALIZE_ARRAY            *
! *    Sets all elements in the arrays            *
! *    to zero.                                    *
! *****

```

```

SUB initialize_array
    MAT xdata1=0
    MAT points=0
    MAT xcor1=0
    MAT xdata2=0
    MAT xcor2=0
    MAT xdata3=0
    MAT xcor3=0
    MAT xdata4=0
    MAT xcor4=0
    MAT xdata5=0
    MAT xcor5=0
END SUB

```

```

! *****
! *                SUB MAKE_xdata1                *
! *    Forms the array xdata1. Finds the          *
! *    difference between each point and its     *
! *    neighbor.                                    *
! *****

```

```

SUB make_xdata1
    LET pointnum=0
    FOR m=1 to total_data_pts
        LET startx=points(m,1)
        FOR j=m+1 to total_data_pts-1
            LET pointnum=pointnum+1
            LET dif1=(startx-points(j,1))*(startx-

```

```

points(j,1))
        LET difsq=sqr(dif1)
        LET xdata1(pointnum,1)=difsq +
xdata1(pointnum,1)
        NEXT j
    NEXT m
END SUB

! *****
! *          SUB make_xcor1          *
! *    Finds the number of distances less *
! *    than r and puts them in the array *
! *    xcor1.                          *
! *****

SUB make_xcor1
    LET count=0
    FOR r=1 to 20 step .2
        LET count = count+1
        LET xcor1(count,1)=r
        LET sumdist=0
        FOR i=1 to 4950
            IF (xdata1(i,1)*10) < r then
                LET sumdist=sumdist + 1
            END IF
        NEXT i
        LET xcor1(count,2)=sumdist/((4950*4949)/2)
    NEXT r
END SUB

! *****
! *          SUB plot_line1          *
! *    Plots the log of the distance r *
! *    vs the log of the number of distances *
! *    less than r. Also calculates *
! *    sums to be used in finding *
! *    the slope in the subroutine *
! *    find_slope.                  *
! *****

SUB plot_line1
    LET old_x=0
    LET old_y=0
    LET old_xy=0
    LET old_xx=0
    FOR i=1 to total_data_pts
        IF xcor1(i,2)<>0 then
            LET xcoord=log(xcor1(i,1)) !r
            LET ycoord=log(xcor1(i,2)*1e7) !C(N,r)
            PLOT xcoord,ycoord
            LET sumx=xcoord+old_x !sum of x coord
            LET old_x=sumx
        END IF
    NEXT i
END SUB

```

```

                LET sumy=ycoord+old_y  !sum of y coord
                LET old_y=sumy
                LET sumxy=xcoord*ycoord+old_xy  !sum of
x*y coord
                LET old_xy=sumxy
                LET sumxx=xcoord*xcoord+old_xx  !sum of
x*x coord
                LET old_xx=sumxx
                LET sumyy=ycoord*ycoord+old_yy  !sum of
y*y coord
                LET old_yy=sumyy
            END IF
        NEXT i
    END SUB

```

```

! *****
! *                SUB find_slope                *
! *    Calculates the slope of a line using      *
! *    linear regression.                        *
! *****

```

```

SUB find_slope
    LET pts=total_data_pts
    LET sxy=sumxy-((sumx*sumy)/((2*pts)-1))
    LET sxx=sumxx-((sumx*sumx)/((2*pts)-1))
    LET syy=sumyy-((sumy*sumy)/((2*pts)-1))
    LET slope=sxy/sxx
    LET r=sxy/(sqr(abs(sxx*syy)))
    PRINT slope
END SUB

```

```

! *****
! *                SUB LINE1                *
! *    Calls the routines to calculate and      *
! *    plot the first line and return the      *
! *    slope.                                  *
! *****

```

```

SUB line1
    CALL make_xdata1
    CALL make_xcor1
    CALL plot_line1
    CALL find_slope
END SUB

```

```

! *****
! *                SUB make_xdata2            *
! *    Forms the array xdata2. Finds the      *
! *    difference between pairs of points.    *
! *****

```

```

SUB make_xdata2

```

```

LET pointnum=0
FOR m=1 to total_data_pts-2
  LET startx=points(m,1)
  LET start2x=points(m+1,1)
  FOR j=m+1 to total_data_pts-2
    LET pointnum=pointnum+1
    LET dif1=(startx-points(j,1))*(startx-
points(j,1))
    LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
    LET difsq=sqr(dif1+dif2)
    LET xdata2(pointnum,1)=difsq +
xdata2(pointnum,1)
  NEXT j
NEXT m
END SUB

```

```

! *****
! *          SUB make_xcor2          *
! *    Finds the number of distances less *
! *    than r and puts them in the array *
! *    xcor2.                          *
! *****

```

```

SUB make_xcor2
  LET count=0
  FOR r=1 to 20 step .2
    LET count = count+1
    LET xcor2(count,1)=r
    LET sumdist=0
    FOR i=1 to 4900
      IF (xdata2(i,1)*10)<r then
        LET sumdist=sumdist + 1
      END IF
    NEXT i
    LET
xcor2(count,2)=(1/((4900*4899)/2))*sumdist
  NEXT r
END SUB

```

```

! *****
! *          SUB plot_line2          *
! *    Plots the log of the distance r *
! *    vs the log of the number of distances *
! *    less than r. Also calculates *
! *    sums to be used in finding *
! *    the slope in the subroutine *
! *    find_slope.                    *
! *****

```

```

SUB plot_line2
  LET old_x=0

```

```

LET old_y=0
LET old_xy=0
LET old_xx=0
FOR i=1 to total_data_pts
  IF xcor2(i,2)<>0 then
    LET xcoord=log(xcor2(i,1)) !r
    LET ycoord=log(xcor2(i,2)*1e7) !C(N,r)
    PLOT xcoord,ycoord
    LET sumx=xcoord+old_x !sum of x coord
    LET old_x=sumx
    LET sumy=ycoord+old_y !sum of y coord
    LET old_y=sumy
    LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
    LET old_xy=sumxy
    LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
    LET old_xx=sumxx
    LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
    LET old_yy=sumyy
  END IF
NEXT i
END SUB

```

```

! *****
! *                SUB LINE2                *
! *    Calls the routines to calculate and    *
! *    plot the second line and return the    *
! *    slope.                                  *
! *****

```

```

SUB line2
  CALL make_xdata2
  CALL make_xcor2
  CALL plot_line2
  CALL find_slope
END SUB

```

```

! *****
! *                SUB make_xdata3           *
! *    Forms the array xdata3. Finds the      *
! *    difference between each three points.  *
! *    neighbor.                              *
! *****

```

```

SUB make_xdata3
  LET pointnum=0
  FOR m=1 to total_data_pts-3
    LET startx=points(m,1)
    LET start2x=points(m+1,1)
    LET start3x=points(m+2,1)

```

```

        FOR j=m+1 to total_data_pts-3
            LET pointnum=pointnum+1
            LET dif1=(startx-points(j,1))*(startx-
points(j,1))
            LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
            LET dif3=(start3x-
points(j+2,1))*(start3x-points(j+2,1))
            LET difsq=sqr(dif1+dif2+dif3)
            LET xdata3(pointnum,1)=difsq +
xdata3(pointnum,1)
        NEXT j
    NEXT m
END SUB

```

```

! *****
! *          SUB make_xcor3          *
! *    Finds the number of distances less *
! *    than r and puts them in the array *
! *    xcor3. *
! *****

```

```

SUB make_xcor3
    LET count=0
    FOR r=1 to 20 step .2
        LET count = count+1
        LET xcor3(count,1)=r
        LET sumdist=0
        FOR i=1 to 4850
            IF (xdata3(i,1)*10)<r then
                LET sumdist=sumdist + 1
            END IF
        NEXT i
        LET
xcor3(count,2)=(1/((4850*4849)/2))*sumdist
    NEXT r
END SUB

```

```

! *****
! *          SUB plot_line3          *
! *    Plots the log of the distance r *
! *    vs the log of the number of distances *
! *    less than r. Also calculates *
! *    sums to be used in finding *
! *    the slope in the subroutine *
! *    find_slope. *
! *****

```

```

SUB plot_line3
    LET old_x=0
    LET old_y=0
    LET old_xy=0

```



```

LET old_xx=0
FOR i=1 to total_data_pts
  IF xcor3(i,2)<>0 then
    LET xcoord=log(xcor3(i,1)) !r
    LET ycoord=log(xcor3(i,2)*1e7) !C(N,r)
    PLOT xcoord,ycoord
    LET sumx=xcoord+old_x !sum of x coord
    LET old_x=sumx
    LET sumy=ycoord+old_y !sum of y coord
    LET old_y=sumy
    LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
    LET old_xy=sumxy
    LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
    LET old_xx=sumxx
    LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
    LET old_yy=sumyy
  END IF
NEXT i
END SUB

```

```

! *****
! *                SUB line3                *
! *    Calls the routines to calculate and    *
! *    plot the third line and return the    *
! *    slope.                                *
! *****

```

```

SUB line3
  CALL make_xdata3
  CALL make_xcor3
  CALL plot_line3
  CALL find_slope
END SUB

```

```

! *****
! *                SUB make_xdata4           *
! *    Forms the array xdata4. Finds the    *
! *    difference between four points.      *
! *****

```

```

SUB make_xdata4
  LET pointnum=0
  FOR m=1 to total_data_pts-4
    LET startx=points(m,1)
    LET start2x=points(m+1,1)
    LET start3x=points(m+2,1)
    LET start4x=points(m+3,1)
    FOR j=m+1 to total_data_pts-4
      LET pointnum=pointnum+1
    
```

```

      LET dif1=(startx-points(j,1))*(startx-
points(j,1))
      LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
      LET dif3=(start3x-
points(j+2,1))*(start3x-points(j+2,1))
      LET dif4=(start4x-
points(j+3,1))*(start4x-points(j+3,1))
      LET difsq=sqr(dif1+dif2+dif3+dif4)
      LET xdata4(pointnum,1)=difsq +
xdata4(pointnum,1)
    NEXT j
  NEXT m
END SUB

```

```

! *****
! *          SUB make_xcor4          *
! *    Finds the number of distances less *
! *    than r and puts them in the array *
! *    xcor4. *
! *****

```

```

SUB make_xcor4
  LET count=0
  FOR r=1 to 20 step .2
    LET count = count+1
    LET xcor4(count,1)=r
    LET sumdist=0
    FOR i=1 to 4800
      IF (xdata4(i,1)*10)<r then
        LET sumdist=sumdist + 1
      END IF
    NEXT i
    LET
xcor4(count,2)=(1/((4800*4799)/2))*sumdist
  NEXT r
END SUB

```

```

! *****
! *          SUB plot_line4          *
! *    Plots the log of the distance r *
! *    vs the log of the number of distances *
! *    less than r. Also calculates *
! *    sums to be used in finding *
! *    the slope in the subroutine *
! *    find_slope. *
! *****

```

```

SUB plot_line4
  LET old_x=0
  LET old_y=0
  LET old_xy=0

```

```

LET old_xx=0
FOR i=1 to total_data_pts
  IF xcor4(i,2)<>0 then
    LET xcoord=log(xcor4(i,1)) !r
    LET ycoord=log(xcor4(i,2)*1e7) !C(N,r)
    PLOT xcoord,ycoord
    LET sumx=xcoord+old_x !sum of x coord
    LET old_x=sumx
    LET sumy=ycoord+old_y !sum of y coord
    LET old_y=sumy
    LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
    LET old_xy=sumxy
    LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
    LET old_xx=sumxx
    LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
    LET old_yy=sumyy
  END IF
NEXT i
END SUB

```

```

! *****
! *                SUB line4                *
! *    Calls the routines to calculate and    *
! *    plot the fourth line and return the    *
! *    slope.                                  *
! *****

```

```

SUB line4
  CALL make_xdata4
  CALL make_xcor4
  CALL plot_line4
  CALL find_slope
END SUB

```

```

! *****
! *                SUB make_xdata5           *
! *    Forms the array xdata5. Finds the      *
! *    difference between each point and its  *
! *    neighbor.                              *
! *****

```

```

SUB make_xdata5
  LET pointnum=0
  FOR m=1 to total_data_pts-5
    LET startx=points(m,1)
    LET start2x=points(m+1,1)
    LET start3x=points(m+2,1)
    LET start4x=points(m+3,1)
    LET start5x=points(m+4,1)

```

```

        FOR j=m+1 to total_data_pts-5
            LET pointnum=pointnum+1
            LET dif1=(startx-points(j,1))*(startx-
points(j,1))
            LET dif2=(start2x-
points(j+1,1))*(start2x-points(j+1,1))
            LET dif3=(start3x-
points(j+2,1))*(start3x-points(j+2,1))
            LET dif4=(start4x-
points(j+3,1))*(start4x-points(j+3,1))
            LET dif5=(start5x-
points(j+4,1))*(start5x-points(j+4,1))
            LET difsq=sqr(dif1+dif2+dif3+dif4+dif5)
            LET xdata5(pointnum,1)=difsq +
xdata5(pointnum,1)
        NEXT j
    NEXT m
END SUB

```

```

! *****
! *           SUB make_xcor5           *
! *   Finds the distance of distances less *
! *   than r and puts them in the array *
! *   xcor5. *
! *****

```

```

SUB make_xcor5
    LET count=0
    FOR r=1 to 20 step .2
        LET count = count+1
        LET xcor5(count,1)=r
        LET sumdist=0
        FOR i=1 to 4750
            IF (xdata5(i,1)*10)<r then
                LET sumdist=sumdist + 1
            END IF
        NEXT i
        LET xcor5(count,2)=sumdist/((4750*4749)/2)
    NEXT r
END SUB

```

```

! *****
! *           SUB plot_line5           *
! *   Plots the log of the distance r *
! *   vs the log of the number of distances *
! *   less than r. Also calculates *
! *   sums to be used in finding *
! *   the slope in the subroutine *
! *   find_slope. *
! *****

```

```

SUB plot_line5

```

```

LET old_x=0
LET old_y=0
LET old_xy=0
LET old_xx=0
FOR i=1 to total_data_pts
  IF xcor5(i,2)<>0 then
    LET xcoord=log(xcor5(i,1)) !r
    LET ycoord=log(xcor5(i,2)*1e7) !C(N,r)
    PLOT xcoord,ycoord
    LET sumx=xcoord+old_x !sum of x coord
    LET old_x=sumx
    LET sumy=ycoord+old_y !sum of y coord
    LET old_y=sumy
    LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
    LET old_xy=sumxy
    LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
    LET old_xx=sumxx
    LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
    LET old_yy=sumyy
  END IF
NEXT i
END SUB

```

```

! *****
! *                SUB line5                *
! *    Calls the routines to calculate and    *
! *    plot the fifth line and return the    *
! *    slope.                                *
! *****

```

```

SUB line5
  CALL make_xdata5
  CALL make_xcor5
  CALL plot_line5
  CALL find_slope
END SUB

```

```

! *****
! *                MAIN PROGRAM                *
! *    Calls subroutines to create each line. *
! *****

```

```

SET COLOR "white"
CALL axis
CALL ticks (1,1) !distance apart on
x and y axis
CALL initialize_array
CALL make_points(x)
CALL line1

```

```
SET COLOR "yellow"  
CALL line2  
SET COLOR "blue"  
CALL line3  
SET COLOR "green"  
CALL line4  
SET COLOR "red"  
CALL line5  
SOUND 300,.5  
END
```

APPENDIX D

COMPUTER PROGRAM "DIM3"

```
! *****
! *                               DIM3                               *
! *   Finds the dimension of the Henon attractor                   *
! *   using Guckenheimer's method.                                 *
! *   Inverse slope of lines plotted approaches                    *
! *   the dimension of the attractor.                              *
! *****
```

```
! *****
! *   Initialize variables and constants. Declare                  *
! *   arrays.                                                       *
! *****
```

```
OPEN #1: printer
SET MODE "egahires"
SET WINDOW 0,8,0,5
DIM points(1000,2)           !# of data pts
DIM xdata1(999,1)
DIM xdata2(998,1)
DIM xdata3(997,1)
DIM xdata4(996,1)
DIM xdata5(995,1)
LET y=.1
LET x=.1                     !initial value
LET a=1.4                    !constant
LET b=.3                     !constant
LET total_data_pts=1000     !# of data pts
```

```
! *****
! *                               SUB AXIS                           *
! *   Draws x and y axis.                                          *
! *****
```

```
SUB axis
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
END SUB
```

```
! *****
! *                               SUB TICKS                               *
! *           Draws ticks on x and y axis.                               *
! *****
```

```
SUB ticks(x,y)                                ! axis with ticks
x and y units apart
  ASK SCREEN u1,u2,v1,v2
  ASK WINDOW x1,x2,y1,y2
  PLOT x1,0;x2,0
  PLOT 0,y1;0,y2
  LET xu=640
  LET p1=abs(v2-v1)*200                        !pixels vertical
  LET p2=abs(u2-u1)*xu                        !pixels horizontal
  LET r=min(p1/50,p2/50)                      !small fraction
  LET d1=r/p1*abs(y2-y1)                      !for x
  LET d2=r/p2*abs(x2-x1)                      !for y
  CALL mark(x1,x,1,d1)
  CALL mark(x2,x,1,d1)
  CALL mark(y1,y,2,d2)
  CALL mark(y2,y,2,d2)
END SUB
```

```
! *****
! *                               SUB MARK                               *
! *           Used by SUB TICKS to draw a single tick.                   *
! *****
```

```
SUB mark(u2,us,c,d)                          !does 1 tick
  IF u2=0 then EXIT SUB
  FOR u=0 to u2 step sgn(u2)*us
    IF c=1 then
      PLOT u,-d;u,d
    ELSE
      PLOT -d,u;d,u
    END IF
  NEXT u
END SUB
```

```
! *****
! *                               SUB FUNCTION                           *
! *           Equations for Henon attractor.                             *
! *****
```

```
SUB function (x)
  LET xx=1-a*x*x+y
  LET yy=b*x
  LET x=xx
  LET y=yy
END SUB
```



```

! *****
! *          SUB MAKE_POINTS          *
! *    Reads x & y values of strange attractor *
! *    equations into an array POINTS *
! *****

```

```

SUB make_points (x)
  FOR i=1 to total_data_pts      !# of data pts
    CALL function (x)
    LET points (i,1)=x
  NEXT i
END SUB

```

```

! *****
! *          SUB INITIALIZE_ARRAY    *
! *    Sets all elements in the arrays *
! *    to zero. *
! *****

```

```

SUB initialize_array
  MAT points=0
  MAT xdata1=0
  MAT xdata2=0
  MAT xdata3=0
  MAT xdata4=0
  MAT xdata5=0
END SUB

```

```

! *****
! *          SUB MAKE_xdata1        *
! *    Forms the array xdata1. Finds the *
! *    distance between each point and its *
! *    neighbor. *
! *****

```

```

SUB make_xdata1
  LET m=1
  LET startx=points(m,1)
  FOR j=m+1 to total_data_pts
    LET dif1=(startx-points(j,1))*(startx-
points(j,1))
    LET difsq=sqr(dif1)
    LET xdata1(j-1,1)=difsq
  NEXT j
END SUB

```

```

! *****
! *          SUB SORT_xdata1        *
! *    Produces a list of distances sorted *
! *    from smallest to largest and puts *
! *    them in xdata1(i,1). *
! *****

```

```

SUB sort_xdata1
  FOR first=1 to total_data_pts-1
    LET smallest=first
    FOR current=first to total_data_pts-1
      IF xdata1(current,1)<xdata1(smallest,1)
then
      LET smallest=current
    END IF
  NEXT current
  LET temp=xdata1(smallest,1)
  LET xdata1(smallest,1)=xdata1(first,1)
  LET xdata1(first,1)=temp
NEXT first
END SUB

```

```

! *****
! *          SUB PLOT_XLINE1          *
! *    Plots the log of the distance   *
! *    vs the log of the number of distances *
! *    less than the distance. Also   *
! *    calculates sums to be used in finding *
! *    the slope in the subroutine     *
! *    FIND_SLOPE.                    *
! *****

```

```

SUB plot_xline1
  LET old_x=0
  LET old_y=0
  LET old_xy=0
  LET old_xx=0
  LET old_yy=0
  FOR i=1 to total_data_pts-1
    IF xdata1(i,1)<>0 then
      LET xcoord=log(xdata1(i,1)*1000)
      LET ycoord=log((i/total_data_pts)*100)
      PLOT xcoord,ycoord
      LET sumx=xcoord+old_x  !sum of x coord
      LET old_x=sumx
      LET sumy=ycoord+old_y  !sum of y coord
      LET old_y=sumy
      LET sumxy=xcoord*ycoord+old_xy  !sum of
x*y coord
      LET old_xy=sumxy
      LET sumxx=xcoord*xcoord+old_xx  !sum of
x*x coord
      LET old_xx=sumxx
      LET sumyy=ycoord*ycoord+old_yy  !sum of
y*y coord
      LET old_yy=sumyy
    END IF
  NEXT i

```

END SUB

```
! *****
! *                SUB LINE1                *
! *    Calls the routines to calculate and   *
! *    plot the line and return the slope.  *
! *****
```

```
SUB line1
  CALL make_xdata1
  CALL sort_xdata1
  CALL plot_xline1
  CALL find_slope
  CALL find_dim
END SUB
```

```
! *****
! *                SUB MAKE_xdata2          *
! *    Forms the array xdata2. Finds the    *
! *    distance between pairs of points.    *
! *****
```

```
SUB make_xdata2
  LET m=1
  LET startx=points(m,1)
  LET startx2=points(m+1,1)
  FOR j=m+1 to total_data_pts-2
    LET dif1=(startx-points(j,1))*(startx-
points(j,1))
    LET dif2=(startx2-points(j+1,1))*(startx2-
points(j+1,1))
    LET difsq=sqr(dif1+dif2)
    LET xdata2(j-1,1)=difsq
  NEXT j
END SUB
```

```
! *****
! *                SUB SORT_xdata2         *
! *    Produces a list of distances sorted  *
! *    from smallest to largest and puts   *
! *    them in xdata2(i,1).               *
! *****
```

```
SUB sort_xdata2
  FOR first=1 to total_data_pts-2
    LET smallest=first
    FOR current=first to total_data_pts-2
      IF xdata2(current,1)<xdata2(smallest,1)
then
          LET smallest=current
      END IF
    NEXT current
```

```

        LET temp=xdata2(smallest,1)
        LET xdata2(smallest,1)=xdata2(first,1)
        LET xdata2(first,1)=temp
    NEXT first
END SUB

! *****
! *                SUB PLOT_xline2                *
! *    Plots the log of the inverse distance      *
! *    vs the log of the number of distances     *
! *    less than the distance. Also              *
! *    calculates sums to be used in finding     *
! *    the slope in the subroutine               *
! *    FIND_SLOPE.                               *
! *****

SUB plot_xline2
    LET old_x=0
    LET old_y=0
    LET old_xy=0
    LET old_xx=0
    LET old_yy=0
    FOR i=1 to total_data_pts-2
        IF xdata2(i,1)<>0 then
            LET xcoord=log(xdata2(i,1)*1000)
            LET ycoord=log((i/total_data_pts)*100)
            PLOT xcoord,ycoord
            LET sumx=xcoord+old_x !sum of x coord
            LET old_x=sumx
            LET sumy=ycoord+old_y !sum of y coord
            LET old_y=sumy
            LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
            LET old_xy=sumxy
            LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
            LET old_xx=sumxx
            LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
            LET old_yy=sumyy
        END IF
    NEXT i
END SUB

! *****
! *                SUB LINE2                *
! *    Calls the routines to calculate and      *
! *    plot the line and return the slope.     *
! *****

SUB line2
    CALL make_xdata2

```

```

CALL sort_xdata2
CALL plot_xline2
CALL find_slope
CALL find_dim
END SUB

! *****
! *          SUB MAKE_xdata3          *
! *    Forms the array xdata3. Finds the *
! *    distance between three points.   *
! *****

SUB make_xdata3
  LET m=1
  LET startx=points(m,1)
  LET startx2=points(m+1,1)
  LET startx3=points(m+2,1)
  FOR j=m+1 to total_data_pts-3
    LET dif1=(startx-points(j,1))*(startx-
points(j,1))
    LET dif2=(startx2-points(j+1,1))*(startx2-
points(j+1,1))
    LET dif3=(startx3-points(j+2,1))*(startx3-
points(j+2,1))
    LET difsq=sqr(dif1+dif2+dif3)
    LET xdata3(j-1,1)=difsq
  NEXT j
END SUB

! *****
! *          SUB SORT_xdata3          *
! *    Produces a list of distances sorted *
! *    from smallest to largest and puts  *
! *    them in xdata3(i,1).              *
! *****

SUB sort_xdata3
  FOR first=1 to total_data_pts-3
    LET smallest=first
    FOR current=first to total_data_pts-3
      IF xdata3(current,1)<xdata3(smallest,1)
then
        LET smallest=current
      END IF
    NEXT current
    LET temp=xdata3(smallest,1)
    LET xdata3(smallest,1)=xdata3(first,1)
    LET xdata3(first,1)=temp
  NEXT first
END SUB

```

```

! *****
! *          SUB PLOT_xline3          *
! *    Plots the log of the inverse distance *
! *    vs the log of the number of distances *
! *    less than the distance. Also *
! *    calculates sums to be used in finding *
! *    the slope in the subroutine *
! *    FIND_SLOPE. *
! *****

SUB plot_xline3
  LET old_x=0
  LET old_y=0
  LET old_xy=0
  LET old_xx=0
  LET old_yy=0
  FOR i=1 to total_data_pts-3
    IF xdata3(i,1)<>0 then
      LET xcoord=log(xdata3(i,1)*1000)
      LET ycoord=log((i/total_data_pts)*100)
      PLOT xcoord,ycoord
      LET sumx=xcoord+old_x !sum of x coord
      LET old_x=sumx
      LET sumy=ycoord+old_y !sum of y coord
      LET old_y=sumy
      LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
      LET old_xy=sumxy
      LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
      LET old_xx=sumxx
      LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
      LET old_yy=sumyy
    END IF
  NEXT i
END SUB

! *****
! *          SUB LINE3          *
! *    Calls the routines to calculate and *
! *    plot the line and return the slope. *
! *****

SUB line3
  CALL make_xdata3
  CALL sort_xdata3
  CALL plot_xline3
  CALL find_slope
  CALL find_dim
END SUB

```

```

! *****
! *          SUB MAKE_xdata4          *
! *    Forms the array xdata4. Finds the *
! *    distance between four points.    *
! *****

```

```

SUB make_xdata4
  LET m=1
  LET startx=points(m,1)
  LET startx2=points(m+1,1)
  LET startx3=points(m+2,1)
  LET startx4=points(m+3,1)
  FOR j=m+1 to total_data_pts-4
    LET dif1=(startx-points(j,1))*(startx-
points(j,1))
    LET dif2=(startx2-points(j+1,1))*(startx2-
points(j+1,1))
    LET dif3=(startx3-points(j+2,1))*(startx3-
points(j+2,1))
    LET dif4=(startx4-points(j+3,1))*(startx4-
points(j+3,1))
    LET difsq=sqr(dif1+dif2+dif3+dif4)
    LET xdata4(j-1,1)=difsq
  NEXT j
END SUB

```

```

! *****
! *          SUB SORT_xdata4          *
! *    Produces a list of distances sorted *
! *    from smallest to largest and puts  *
! *    them in xdata4(i,1).              *
! *****

```

```

SUB sort_xdata4
  FOR first=1 to total_data_pts-4
    LET smallest=first
    FOR current=first to total_data_pts-4
      IF xdata4(current,1)<xdata4(smallest,1)
then
        LET smallest=current
      END IF
    NEXT current
    LET temp=xdata4(smallest,1)
    LET xdata4(smallest,1)=xdata4(first,1)
    LET xdata4(first,1)=temp
  NEXT first
END SUB

```

```

! *****
! *          SUB PLOT_xline4          *
! *    Plots the log of the inverse distance *
! *    vs the log of the number of distances *
! *    less than the distance. Also *
! *    calculates sums to be used in finding *
! *    the slope in the subroutine *
! *    FIND_SLOPE. *
! *****

SUB plot_xline4
  LET old_x=0
  LET old_y=0
  LET old_xy=0
  LET old_xx=0
  LET old_yy=0
  FOR i=1 to total_data_pts-4
    IF xdata4(i,1)<>0 then
      LET xcoord=log(xdata4(i,1)*1000)
      LET ycoord=log((i/total_data_pts)*100)
      PLOT xcoord,ycoord
      LET sumx=xcoord+old_x !sum of x coord
      LET old_x=sumx
      LET sumy=ycoord+old_y !sum of y coord
      LET old_y=sumy
      LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
      LET old_xy=sumxy
      LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
      LET old_xx=sumxx
      LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
      LET old_yy=sumyy
    END IF
  NEXT i
END SUB

! *****
! *          SUB LINE4          *
! *    Calls the routines to calculate and *
! *    plot the line and return the slope. *
! *****

SUB line4
  CALL make_xdata4
  CALL sort_xdata4
  CALL plot_xline4
  CALL find_slope
  CALL find_dim
END SUB

```



```

! *****
! *           SUB MAKE_xdata5           *
! *       Forms the array xdata5. Finds the *
! *       distance between five points.    *
! *****

```

```

SUB make_xdata5
  LET m=1
  LET startx=points(m,1)
  LET startx2=points(m+1,1)
  LET startx3=points(m+2,1)
  LET startx4=points(m+3,1)
  LET startx5=points(m+4,1)
  FOR j=m+1 to total_data_pts-5
    LET dif1=(startx-points(j,1))*(startx-
points(j,1))
    LET dif2=(startx2-points(j+1,1))*(startx2-
points(j+1,1))
    LET dif3=(startx3-points(j+2,1))*(startx3-
points(j+2,1))
    LET dif4=(startx4-points(j+3,1))*(startx4-
points(j+3,1))
    LET dif5=(startx5-points(j+4,1))*(startx5-
points(j+4,1))
    LET difsq=sqr(dif1+dif2+dif3+dif4+dif5)
    LET xdata5(j-1,1)=difsq
  NEXT j
END SUB

```

```

! *****
! *           SUB SORT_xdata5           *
! *       Produces a list of distances sorted *
! *       from smallest to largest and puts *
! *       them in xdata5(i,1).             *
! *****

```

```

SUB sort_xdata5
  FOR first=1 to total_data_pts-5
    LET smallest=first
    FOR current=first to total_data_pts-5
      IF xdata5(current,1)<xdata5(smallest,1)
then
        LET smallest=current
      END IF
    NEXT current
    LET temp=xdata5(smallest,1)
    LET xdata5(smallest,1)=xdata5(first,1)
    LET xdata5(first,1)=temp
  NEXT first
END SUB

```

```

! *****
! *                SUB PLOT_xline5                *
! *    Plots the log of the inverse distance      *
! *    vs the log of the number of distances     *
! *    less than the distance. Also              *
! *    calculates sums to be used in finding     *
! *    the slope in the subroutine               *
! *    FIND_SLOPE.                               *
! *****

```

```

SUB plot_xline5
  LET old_x=0
  LET old_y=0
  LET old_xy=0
  LET old_xx=0
  LET old_yy=0
  FOR i=1 to total_data_pts-5
    IF xdata5(i,1)<>0 then
      LET xcoord=log(xdata5(i,1)*1000)
      LET ycoord=log((i/total_data_pts)*100)
      PLOT xcoord,ycoord
      LET sumx=xcoord+old_x !sum of x coord
      LET old_x=sumx
      LET sumy=ycoord+old_y !sum of y coord
      LET old_y=sumy
      LET sumxy=xcoord*ycoord+old_xy !sum of
x*y coord
      LET old_xy=sumxy
      LET sumxx=xcoord*xcoord+old_xx !sum of
x*x coord
      LET old_xx=sumxx
      LET sumyy=ycoord*ycoord+old_yy !sum of
y*y coord
      LET old_yy=sumyy
    END IF
  NEXT i
END SUB

```

```

! *****
! *                SUB LINE5                *
! *    Calls the routines to calculate and      *
! *    plot the line and return the slope.     *
! *****

```

```

SUB line5
  CALL make_xdata5
  CALL sort_xdata5
  CALL plot_xline5
  CALL find_slope
  CALL find_dim
END SUB

```

```

! *****
! *          SUB FIND_SLOPE          *
! *    Calculates the slope of a line using *
! *    linear regression, and the correlation *
! *    coefficient, r.                  *
! *****

```

```

SUB find_slope
  LET pts=total_data_pts-1
  LET sxy=sumxy-((sumx*sumy)/((2*pts)-1))
  LET sxx=sumxx-((sumx*sumx)/((2*pts)-1))
  LET syy=sumyy-((sumy*sumy)/((2*pts)-1))
  LET slope=sxy/sxx
  LET r=sxy/(sqr(sxx*syy))
END SUB

```

```

! *****
! *          SUB FIND_DIM          *
! *    Calculates the dimension as the *
! *    inverse of the slope.          *
! *****

```

```

SUB find_dim
  LET dimension=1/slope
  PRINT "s=",slope;
  PRINT "d=",dimension;
END SUB

```

```

! *****
! *          MAIN PROGRAM          *
! *    Calls subroutine to create line. *
! *****

```

```

SET COLOR "white"
CALL axis
CALL ticks(1,1)
CALL initialize_array
CALL make_points(x)
CALL line1
SET COLOR "yellow"
CALL line2
SET COLOR "red"
CALL line3
SET COLOR "blue"
CALL line4
SET COLOR "green"
CALL line5
SOUND 300,.5
END

```