1-D MAP FOR THE DOUBLE SCROLL FAMILY

by

Leon O. Chua and Irene Tichonicky

Memorandum No. UCB/ERL M89/108

28 August 1989

1-D MAP FOR THE DOUBLE SCROLL FAMILY

by

Leon O. Chua and Irene Tichonicky

Memorandum No. UCB/ERL M89/108

28 August 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

1-D MAP FOR THE DOUBLE SCROLL FAMILY

by

Leon O. Chua and Irene Tichonicky

Memorandum No. UCB/ERL M89/108

28 August 1989

1-D MAP FOR THE DOUBLE SCROLL FAMILY

Leon O. Chua[†] *and Irene Tichonicky*[††]

ABSTRACT

We use the 1-D map, a simplification of the two-dimensional Poincaré half-return maps, to study periodic windows of the Double Scroll, a strange attractor observed in Chua's circuit.

First, with a generalization of Kneading Theory, we determine the form and the order of appearance of periodic orbits in the 1-D map. With this information, we then find the form and the period of the corresponding orbits of the Double Scroll.

† Leon O. Chua is with the Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley.

† † Irene Tichonicky if affiliated with the Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France.

Contents :

1. Introduction

This paper gives an analytical classification of the periodic orbits of the Double-Scroll Family [1].

1.1. The Double Scroll

The Double Scroll [1] is a strange attractor which can be observed in Chua's circuit, a simple autonomous electrical circuit comprising four linear circuit elements (one resistor, one inductor and two capacitors) and a three-segment piecewise-linear resistor. Chua's circuit is shown in figure 1.

The behavior of this circuit is governed by a set of three ordinary differential equations :

$$C_1 \frac{dv_{C_1}}{dt} = G(v_{C_2} - v_{C_1}) - g(v_{C_1})$$

$$C_2 \frac{dv_{C_2}}{dt} = G(v_{C_1} - v_{C_2}) + i_L$$

$$L \frac{di_L}{dt} = -v_{C_2}$$

where the capacitor voltages v_{C_1} and v_{C_2} and the inductor current i_L are chosen as state variables. C_1 and C_2 are the capacitances shown in fig. 1 ; L is the inductance, and G the value of the conductance.

The symmetric nonlinear resistor (figure 2) is described by the piecewise-linear equation :

$$i_R = g(v_R) = m_0 v_R + \frac{1}{2}(m_1 - m_0)|v_R - B_p|$$

where m_0 and m_1 are the slopes in the outer and inner regions and B_p is the breakpoint [1].

For a particular set of values (m_0=–0.5, m_1=–0.8, B_p=1, $1/C_1$=9, $1/C_2$=1, $1/L$=7 and G=0.7), a chaotic attractor has been observed [2] (figure 3).

Via the following rescaling :

$$x = \frac{v_{C_1}}{B_p} \quad y = \frac{v_{C_2}}{B_p} \quad z = \frac{i_L}{B_p\, G}$$

$$\theta = t\,\frac{G}{C_2}$$

$$a = \frac{m_1}{G} \quad b = \frac{m_0}{G}$$

$$\alpha = \frac{C_2}{C_1} \quad \beta = \frac{C_2}{LG^2}$$

the equations of the circuit can be rewritten in the following dimensionless form :

$$\frac{dx}{d\theta} = \alpha \left[y - (b+1)x - \frac{1}{2}(a-b)(|x+1| - |x-1|) \right]$$

$$\frac{dy}{d\theta} = x - y + z$$

$$\frac{dz}{d\theta} = -\beta y$$

The Double-Scroll system is therefore described by a set of four parameters {α, β, a, b}.

The geometric structure of the Double-Scroll attractor is shown figure 4.

1.2. Poincaré map :

To analyze chaotic phenomena in this system, it is useful to consider the two-dimensional Poincaré half return maps derived in [3]. Half-return maps cannot in general be calculated by an explicit formula because the coordinates of the return points can only be found by solving a pair of transcendental equations. Nevertheless, the points which describe the qualitative behavior of the circuit lie on an infinitesimally-narrow corridor when the real eigenvalue in the outer region of the system is relatively large compared to the other eigenvalues [3]. Therefore, it is possible to construct a one-dimensional Poincaré map (1-D map) for many parameter values. This one-dimensional map has an explicit formula.

A previous study [3] has shown that the 1-D map accurately predicts all of the qualitative behavior and that there is a correspondence between the periodic points of the 1-D map and the periodic orbits in the Double Scroll system. In particular, a stable periodic point of the 1-D map corresponds to a stable periodic orbit and an unstable periodic point to a saddle-type periodic orbit of the Double Scroll.

Another study [4] has shown that the definition of the 1-D map can be extended. The extension yields a well-defined piecewise-continuous function, which we shall use (figure 5).

1.3. General results for the Double Scroll circuit

The figure 6 from [4] shows the general behavior of the Double Scroll for {a, b} fixed.

For most fixed values of β, and with α increasing, the qualitative bifurcation structure is indepen-

dent of β. In the following, we will study the birth and death of periodic points of the extended 1-D map for a fixed value of $\beta (= 15)$, and with α increasing. For each value of α we obtain a new map denoted f_α.

2. 1-D map - general behavior

The evolution of the 1-D map for β fixed and α increasing shows that it is possible to divide the interval of definition of α in six parts :

1. 1-D map unimodal (the function has an invariant interval with one critical point, a maximum)
2. 1-D map multimodal
3. first homoclinicity of period 2
4. homoclinicity of period 1
5. second homoclinicity of period 2
6. death of periodic orbits

See figures 7 through 13.

3. 1-D map unimodal

A continuous map f is unimodal if :

- $f : I = [-a, a] \rightarrow I = [-a, a]$
- f has a single critical point (point where the derivative is equal to 0) in I.

The 1-D map has several extrema, but it is possible to find an invariant interval where there is just one maximum and the images of every points are in this interval after one iteration. The 1-D map is unimodal when the image of the first maximum is less than or equal to the abscissa of the first minimum. (For $\beta = 15.$, we have f(maxi) = mini when $\alpha = 9.10472808$).

To describe the dynamics of the 1-D map f_α, we want to classify the periodic orbits. Sarkovskii's theorem [5] gives the order of first appearance of orbits with particular periods :

if $f : \mathbb{R} \rightarrow \mathbb{R}$ is a continuous map which has a periodic orbit of period n, then f has an orbit of

period m for each m in $\mathbb{N}$ and n < m, where < is the order :

$$3 < 5 < 7 < .... < 2\times 3 < 2\times 5 < .... < 2^l 3 < 2^l 5 < .... < 2^l < .... < 2^2 < 2 < 1.$$

In this work, we use a generalization of kneading theory (see Appendix for more details). We define the *kneading invariant*, the semi-infinite sequence by :

$$\nu(f)=(\nu_n)_{n=0}^{\infty}=\lim_{x\uparrow 1^{st}\max}\theta(x)$$

In our case the function $\alpha\rightarrow\nu(f_\alpha)$ is decreasing.

The 1-D map is regular : the only bifurcations of periodic points are period-doubling and saddle-node bifurcations [3].

Using this theory, it is possible to order the set of kneading invariants. Holmes and Whitley [9] showed that the map has an attracting periodic orbit if and only if $\nu(f)$ is periodic.

We observe that if $\nu(f)$ is periodic with period n, then the 1-D map has a periodic orbit of period n and $0\le(f^n)'(x)\le 1$ while if $\nu(f)$ is antiperiodic then f has a periodic orbit which has either period n and $-1\le(f^n)'(x)<0$ or period 2n and $0\le(f^{2n})'(x)\le 1$.

Let's define the following notation : if β is a finite block of -s and +s. β' is the sequence obtained by iterating β indefinitely and $\overline{\beta}$ the sequence obtained by changing all the signs of β.

First we have the birth of a period-1 orbit and ν = (+)' does not change until the fixed point crosses the first maximum. Then ν becomes (+ -)'. α still increases and the stable periodic orbit becomes unstable and creates two period-2 (one stable and one unstable) periodic orbits in a period-doubling bifurcation. But ν is equal to (+ -)' until one of the period-2 points crosses the critical point and $\nu(f)$ becomes (+ - - +)'. The first elements of the set of the kneading invariants are :

(+)'
(+ -)'
(+ - - +)'
(+ - - + - + + -)'.

In fact, in general, we have the same type of behavior for all periodic kneading invariants. If $\nu(f)$ is any admissible kneading invariant, $\nu(f)$ * (+ -)' is the next lower admissible kneading invariant.

The operation * is defined as follows : for any sequence $\nu = (\nu_n)_{n=0}^{\infty}$ of period m and an admissible $\gamma = (\gamma_n)_{n=0}^{\infty}$

$$\nu * \gamma = (\theta_n)_{n=0}^{\infty} \text{ where :}$$

$$\theta_{mk+i} = \gamma_k \nu_i, \quad 0 \le i < m$$

4. 1-D map multimodal

In this case, the invariant interval has several minima and maxima (this occurs when the image of the first maximum is greater than the first minimum). We apply kneading theory again but we here use new sequences (see Appendix - continuous maps), because for multimodal maps it is possible to have several coexisting periodic orbits of different period. For example, when $\alpha = 9.337$, there are two stable periodic orbits (one of period 2 and one of period 3). The pictures of the 1-D map iterated 2 and 3 times show this (figures 14 and 15). In this case the dynamical representation of the first maximum of f_α is : (+ - -)' and the dynamical representation of the third maximum is : (+ +)'. Therefore, it is necessary to look not only at the first maximum but at all the maxima.

To determine the period of the kneading invariant the definition with + and - does not give enough information : it is impossible, for $\alpha = 9.12136$, to find the period of $\nu_1(f_\alpha) = + + + + + \ldots$ (there is more than one interval where the function is increasing or decreasing). For each extremum, we also compute the sequence $\mu(i)$.

With these definitions we obtain similar results as for the unimodal 1-D map. When a periodic orbit is created, one of the maxima has a periodic kneading invariant $(\beta)'$ and $1 \ge (f^n)'(x) \ge 0$ at this point. The kneading invariant is constant until $-1 \le (f^n)'(x) \le 0$. At this moment, it is equal to $(\beta\bar{\beta})'$. As we increase α still further, the periodic orbit becomes unstable and creates a periodic orbit of period 2n, the same maximum has a kneading invariant equal to $(\beta\bar{\beta})'$, and the behavior restarts.

Therefore, for any periodic kneading invariant of any maximum $\nu_i(f_\alpha)$, the next periodic kneading invariant of the same maximum is $\nu_i(f_\alpha)^*$ (+ -)'. In this case it is more difficult to classify the kneading invariants : the function $\alpha \to \nu_i(f_\alpha)$ is not decreasing.

For $\alpha = 9.07656$, $\nu_1(f_\alpha) = (+ - - - -)'$.

For $\alpha = 9.12137$, $\nu_1(f_\alpha) = (+ + + + +)'$.

When α increases, we have more and more intervals where the function is monotone. The new stable periodic orbit always has points in the newly created interval. Therefore, the kneading invariant changes since it is based on the itinerary of the stable periodic orbit.

It is still possible to order the sequence $\mu(i)$. For each fixed i, the function $\alpha \rightarrow \mu(i)$ is increasing (each extremum is considered independently).

Suppose that α_b is a bifurcation point and that there is a new stable period-k orbit. The kneading invariant $(\beta)'$ is periodic of period k ; the itinerary A(i) is periodic with the same period and the sequence $\mu(i)$ is periodic of period k or 2k. The sequence changes when the sign of $(f^k)'$ (max i) passes from + to -. The next sequence $\mu'(i)$ has the same first terms as the sequence $\mu(i)$ and $\mu'_{k+1}(i) = \mu_{k+1}(i) + 1$. For period 2, we have :

$$\alpha = 9.333 \quad \mu(5) = 5\ 1\ 5\ 1...$$
$$\alpha = 9.335 \quad \mu(5) = 5\ 1\ 6\ -1\ -6\ 1\ 6...$$

This result can be explained by the usual kneading theory : after $(\beta)'$, the next kneading invariant is $(\beta\bar{\beta})'$. The term (k + 1) changes from + to -, so $\mu(i)$ also changes. With this result, it is possible to reconstruct the next itinerary A'(i) because it is periodic with period k (after the first term) and we know the first (k + 1) terms of $\mu(i)$. Like the kneading invariants, A'(i) is constant even when the stable periodic orbit of period k becomes unstable and creates one stable and one unstable periodic orbit, each of period 2k. The previous method will give the next itinerary.

To characterize each periodic orbit we use the value of α, the corresponding periodic kneading invariant $\nu_i(f_\alpha)$, μ of the same maximum, and the number of this maximum.

We also observe the extinctions of periodic orbits for α increasing. We have found the value of α where the periodic orbit is stable before its disappearance. It is obvious that if α is increasing we have the birth of periodic orbits instead of their deaths. We observe the same succession of kneading invariants as previously if we consider the minima and α decreasing instead of the maxima with α increasing. That explains why we changed the limit of the minima. It is also possible to find the next (when α decreases) sequence $\mu(i)$. In this case α is decreasing, as is $\mu(i)$. Consequently, if $\mu(i)$ is the itinerary of the minimum i and if the 1-D map has a periodic orbit of period k with $0 \leq (f^n)'(min\ i) \leq 1$, the next

itinerary $\mu'(i)$ satisfies $\mu'_{k+1}(i)=\mu_{k+1}(i)-1$.

For period 3 :

$$\alpha = 9.8617 \qquad \mu(2) = 3\ (1\ 1\ 2\ -1\ -1\ -2)'$$
$$\alpha = 9.8618 \qquad \mu(2) = 3\ (1\ 1\ 3)'$$

5. Results.

Table of the bifurcation points for $\beta = 15.$, $m_0 = -0.14$, $m_1 = 0.28$. The passage of one extremum is represented by ---.

The line _._ means that there is continuation of the usual period-doubling sequence.

α	period	kn.invariant	itinerary	#extremum
7.4	1	(+)'	(1)'	1
8.5	2	(+ -)'	1(22)'	1
8.73	4	(+ - - +)'	1(2122)'	1
...	...	...	...	...
8.866	5	(+ - - + -)'	1(21222)'	1
8.917	3	(+ - -)'	1(212)'	1
8.9734	5	(+ - - - +)'	1(21121)'	1
9.0215	4	(+ - - -)'	1(2112)'	1
9.0765	5	(+ - - - -)'	1(21112)'	1
---	---	---	---	---
9.12136	5	(+ + + + +)'	1(31111)'	1
9.1398	4	(+ + + +)'	1(3111)'	1
9.147534	5	(+ + + + -)'	1(31122)'	1
9.1574	3	(+ + +)'	1(311)'	1
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---
9.17729	5	(+ + + - +)'	1(71121)'	1
9.186555	3	(+ + +)'	1(711)'	1
9.204255	4	(+ + + -)'	1(7122)'	1
---	---	---	---	---
9.29585	4	(+ - - +)'	1(8121)'	1
9.333	2	(+ +)'	7(17)'	7
9.3371	3	(+ - -)'	1(812)'	1
9.34	4	(+ + - -)'	7(1818)'	7
9.463216	3	(+ + +)'	7(117)'	6
---	---	---	---	---
9.72	2	(+ +)'	3(13)'	3
9.8618	3	(+ + +)'	3(113)'	2
10.065	3	(+ + +)'	1(911)'	1
10.17367	2	(+ +)'	7(17)'	6
10.42	2	(+ +)'	1(91)'	1
10.548495	3	(+ + -)'	3(122)'	2
10.58545	3	(+ + -)'	1(922)'	1
10.598211	2	(+ -)'	7(28)'	7
10.612274	2	(+ +)'	3(13)'	2
10.618951	3	(+ + +)'	3(193)'	3
10.8154	1	(+)'	7(7)'	7
10.818	2	(+ -)'	7(87)'	7
10.819	3	(+ - -)'	7(877)'	7
10.923	3	(+ + +)'	1(991)'	1
12.24449	1	(+)'	3(3)'	2

6. Periodic orbits of the Double Scroll

6.1. Introduction

The previous paragraphs give a method for characterizing the periodic orbits of the 1-D map. We know that, for each stable periodic orbit of the 1-D map, there is a stable periodic orbit in the Double Scroll system, but the periods of these orbits are not always equal.

If we look at the geometric structure of the Double Scroll (fig. 4), we can see that it is symmetric with respect to the origin. The behavior in the upper and the lower region is identical ; the middle region is different. It is therefore possible to split the dynamics between two half-spaces. The 1-D map considers only the upper region. Therefore, by symmetry, for each fixed point x of the 1-D map there are two points in the Double Scroll : -x and x. In figure 4, we see also that some trajectories stay in the upper region and others transfer : there are two different corridors. It is also possible to find the limit of the two corridors in the 1-D map (it is represented by the horizontal and the vertical line in figure 5). Chua, Komuro and Matsumoto [3] show that there is a point a such that if the fixed point is between 0 and a, the next fixed point in the trajectory will be greater than 0 and if the fixed point is greater than a, the next fixed point will be lower than 0. For example, for period 1 and $0<x_1<a$ fig 16 a) shows the periodic orbit of the Double Scroll and fig 16 b) a more simple picture. In this case, there are two period-1 orbits. If $x_1>a$ there is one period-1 orbit (see figure 17).

For period n, there are n fixed points in the 1-D map :

$$0 < x_1 < \ldots < x_n < \infty$$

The type of the period-n orbit of the 1-D map is uniquely characterized by the permutation of the indices {2, ...,n}. The type of the periodic orbits of the Double Scroll system is therefore determined by the position of a in the sequence $\{0, x_1, \ldots, x_n\}$. The maximum number of distinct types of period-n orbits of the Double Scroll system is equal to :

$$N = \frac{(n+1)!}{n}$$

6.2. Form of the periodic orbits

The periodic orbits of the Double Scroll system are unknotted knots ; they are equivalent to the trivial knot (see figure 18). It is possible to obtain every periodic orbit in the Double Scroll system from those of period 1 (see figures 16 and 17) by making twists.

6.3. Determination of the period

It is possible with the itineraries of the maxima of the 1-D map, to determine the period of the corresponding orbit of the Double Scroll system.

When the 1-D map has a periodic orbit of period n, the itinerary of one maximum a_0 (A)' is periodic after the first term. The length of A is n. A is also the itinerary of the periodic orbit ; it gives the intervals where each fixed point is. First, we have to know how many periodic points are greater than the separation point a. We observe that a is always in the fifth interval. Therefore, m, the number of fixed points greater than a, is equal to the number of elements of A greater than five. With the pair {n,m}, we can complete the following table. It gives for each pair {n,m} the period of the periodic orbit of the Double Scroll system.

n \ m	0	1	2	3	4	5	6	7
1	1	1						
2	2	3	1					
3	3	5	2	3				
4	4	7	3	5	2			
5	5	9	4	7	3	5		
6	6	11	5	9	4	7	3	

For example, for $\alpha = 8.5$, the 1-D map has a stable periodic orbit of period 2, both fixed points are lower than a (see the table of the previous paragraph) and the Double Scroll system has a periodic orbit of period 2 (see figure 18 (a)). In this case $n = 2$, $m = 0$, and $p = 2$. For $\alpha = 9.3371$, the 1-D map

has a stable periodic orbit of period 3 (n = 3), one fixed point is greater than a (m = 1), therefore the Double Scroll has a periodic orbit of period 5. The simulation verifies this result (see figure 18 (b)).

Let's see how to complete a line. Suppose that we know the periods for a periodic orbit of the 1-D map of period (n-1) and that we want to know the behavior of the periodic orbit of the 1-D map of period n. The number in the box of the first column is equal to one more than the previous number of the same column. To find the number in the second column, we have to add 2 to the previous number of the same column, for the third column add 1, for the fourth column add 2 ... until we reach the last box (column where the number m of fixed points greater than a is equal to the period n). The number in this box is equal to n if n is odd or to n/2 if n is even. In general, the period is equal to $\frac{2n - m}{2}$ if m is even and to $2n - m$ if m is odd. So for period 6 :

first column, m = 0 : 5 + 1 = 6

second column, m = 1 : 9 + 2 = 11

third column, m = 2 : 4 + 1 = 5

fourth column, m = 3 : 7 + 2 = 9

fifth column, m = 4 : 3 + 1 = 4

sixth column, m = 5 : 5 + 2 = 7

seventh column, m = n = 6 : 6/2 = 3

Conclusion

The study of the 1-D map shows that, although it is an approximation of the two-dimensional Poincaré map, it gives many results about the periodic windows of the Double Scroll system.

In the future, it would be interesting to find a method other than knot theory for classifying the periodic orbits of the Double Scroll system and to determine what other insights the 1-D map can provide.

Appendix : Kneading Theory - definitions

Kneading theory is an elaborate version of symbolic dynamics [8]. It gives information about the orbits of critical points.

First we will consider a unimodal function. We then generalize this theory to continuous maps with a finite number of critical points.

1. Unimodal maps

f, a continuous map, is unimodal if :

- $f : I = [-a, a] \to I$
- f has a single critical point (maximum) at $x = 0$.

We only deal with continuous maps of the interval [-a, a] into itself. The results breakdown when the continuity assumption is violated. The choice of an interval [-a, a] is of course arbitrairy, since the change of coordinate $x = a\,\frac{2x' - (b+d)}{d-b}$ will transform [b, d] -> [b, d] into [-a, a] -> [-a, a]. We consider maps with one maximum and we assume that this maximum occurs for $x = 0$. This can sometimes be achieved by a differentiable coordinate transformation [10].

For unimodal maps, we conjecture that the Schwarzian derivative $\frac{f'''(x)}{f'(x)} - \frac{3}{2}\left[\frac{f''(x)}{f'(x)}\right]^2$ is negative on I-{0}. Singer [6] shows that if x is a stable periodic point of f, then there is either a critical point of f or an endpoint of I whose ω-limit set is the orbit of x. In our case $|f'(-a)| > 1$ and $|f'(a)| > 1$, these two points are unstable, so the map has at most one stable periodic orbit.

The *itinerary* of x in I is the semi-infinite sequence $\varepsilon(x)=(\varepsilon_n(x))_{n=0}^{\infty}$.

$$\varepsilon_n(x) = \begin{cases} +1 & \text{if } f^n(x) \text{ is in } [-a,0[\\ 0 & \text{if } f^n(x)=0 \\ -1 & \text{if } f^n(x) \text{ is in }]0,a] \end{cases}$$

The mapping x -> ε(x) does not indicate the ordering of the interval because f reverses the orientation on]0,a]. Let's take two points x and y in]-a, 0[(figure 19). In this case $\varepsilon_0(x) = 1$ and $\varepsilon_0(y) = 1$. f is increasing on]-a, 0[so $f(x) < f(y)$. f(x) and f(y) are greater than 0 so $\varepsilon_1(x) = -1$ and $\varepsilon_1(y) = -1$. f

is decreasing on]0, a[the next iteration reverses the order of (x, y) : $f^2(x) > f^2(y)$. For the next iterations : $\varepsilon_2(x) = 1$, $\varepsilon_2(y) = 1$ and $\varepsilon_3(x) = 1$ and $\varepsilon_3(y) = 1$ (see figure). But $f^2(x) > f^2(y)$ and $f^3(x) > f^3(y)$. Therefore ε_2 and ε_3 do not give any information about the variations of f^2, f^3, f^4. To study the dynamics, it is useful to know the sign of $(f^n)'$ in a neighborhood of x. Consequently we associate with x *the invariant coordinate* $\theta(x)=(\theta_n(x))_{n=0}^{\infty}$ by $\theta_n(x) = \prod_{i=0}^{i=\infty}\varepsilon_i(x)$, or equivalently :

$$\theta_n(x) = \begin{cases} +1 & \text{if } f^{n+1} \text{ is orientation preserving near x} \\ 0 & \text{if } f^m(x)=0 \text{ for some m with } 0 \le m \le n \\ -1 & \text{if } f^{n+1} \text{ is orientation reversing near x} \end{cases}$$

The ordering of the sequence $\theta(x)$ is defined by :

$$\theta(x) < \theta(y)$$
$$\text{if, for } m < n,\ \theta_m(x) = \theta_m(y) \text{ and } \theta_n(x) < \theta_n(y).$$

$\theta : x \to \theta(x)$ is monotone decreasing. Let's take two points in I : $x < y$. We suppose that, for $m < n$, $\theta_m(x) = \theta_m(y)$ and $\theta_n(x) \neq \theta_n(y)$.

- if $\theta_{n-1}(x) = -1$, f^n is orientation reversing, $f^n(x) > f^n(y)$. By hypothesis, $\theta_n(x)$ and $\theta_n(y)$ are different and so are ε_n. Because of the inequality of f^n we assume that $\varepsilon_n(x) = -1$ and $\varepsilon_n(y) = 1$. Therefore, $\theta_n(x) = 1$ and $\theta_n(y) = -1$. Hence $\theta(x) > \theta(y)$.
- if $\theta_{n-1}(x) = 1$, f^n is orientation preserving, $f^n(x) < f^n(y)$. In this case $\varepsilon_n(x) = 1$ and $\varepsilon_n(y) = -1$, $\theta_n(x) = 1$ and $\theta_n(y) = -1$. Therefore, $\theta(x) > \theta(y)$.

We define the *kneading invariant* $\nu(f) = (\nu_n)_{n=0}^{\infty}$ by

$$\nu_n(f) = \lim_{x \uparrow 0} \theta_n(x)$$

The monocity of the invariant coordinate implies that for each x in I either $\theta_n(x)=0$ for all $n \ge 0$, $\theta(x) \ge \nu(f)$ or $\theta(x) \le -\nu(f)$.

We say that a sequence θ is $\nu(f)$*–admissible* if for all $n \ge 0$

- either $\theta_k(x) = 0$ for all $k \ge n$

- or $|\sigma^n(\theta)| \geq \nu(f)$ where σ^n is the term number n of the sequence and $|(\theta_n)_{n=0}^{\infty}| = (\theta_0 \theta_n)_{n=0}^{\infty}$

Then by construction, $\theta(x)$ is $\nu(f)$-admissible for each x in I, and so $\theta(x^+) = \lim_{y \downarrow x} \theta(y)$ and $\theta(x^-) = \lim_{y \uparrow x} \theta(y)$. Conversely, Guckenheimer [7] has shown that : if θ is a $\nu(f)$-admissible sequence, there is a point x in I such that θ is either $\theta(x)$, $\theta(x^+)$ or $\theta(x^-)$.

Therefore, $\nu = \nu(f) = \theta(0^-)$ is also $\nu(f)$-admissible.

Any sequence ν that satisfies $|\sigma^n(\nu)| \geq \nu$ is called an *admissible kneading invariant*.

ν is *periodic* with period n if $\nu_{n+i} = \nu_i$ for all $i > 0$ and *antiperiodic* with period n if $\nu_{n+i} = -\nu_i$ for all $i > 0$.

2. Continuous maps

Now we consider f : I -> I a continuous map of the interval $[c_0, c_l]$ with a finite number of turning points $c_1 < \cdots < c_{l-1}$.

For each point x in I we define

- the *address* of a point x is the j if x is in $] c_j , c_{j+1} [$ and the symbol c_j if $x = c_j$.
- the itinerary of a point x is the sequence $A(x) = \{A_0(x), A_1(x), ..\}$ of the addresses of the points x, f(x),..
- the infinite sequence $(\varepsilon_n(x))_{n=0}^{\infty}$, where for x in I_i

$$\varepsilon_n = \begin{cases} +1 & \text{if } f^n \text{ is increasing on } I_i \\ -1 & \text{if } f^n(x) \text{ is decreasing on } I_i \end{cases}$$

- $\theta_n = \prod_{j=0}^{j=n} \varepsilon_n$ the *kneading invariant* $\nu_i(f)$ of each extremum i :
  - for a maximum, $\lim_{x \uparrow \max i} \theta(x) = \nu_i(f)$
  - for a minimum, $\lim_{x \downarrow \min i} \theta(x) = \nu_i(f)$
- the infinite sequence $\mu(x) = (\mu_n(x))_{n=0}^{\infty}$ where :

$$\mu_n(x) = \prod_{i=0}^{i=n-1} \varepsilon(A_i(x)) A_n(x)$$

and $\mu(i) = \lim_{x \uparrow \max i} \mu(x)$ or $\mu(i) = \lim_{x \downarrow \min i} \mu(x)$

Program description

1. Users Manual

The name of the program is ODM4.

This program plots the 1-D map, for user-specified values of m_0, m_1, α, β. There are various options : determination of periodic points, points of bifurcation, addresses, and the symbolic dynamics of the extrema, picture in α–β space of the periodic points.

1. Type *odm4* or *odm4 filename* if you want to keep the results in a file.
2. Choose the parameters values of m_0, m_1, α, β. The program maps the 1-D map for the chosen parameters.
3. Options
   - ? : list of the available options.
   - a_beta : gives in α–β space the regions where there are periodic points of period n. The user chooses the boundaries of α (alphamin, alphamax) and β (bmin, bmax), and the period. The program divides each interval in 20 parts and looks for each value of the pair (α,β) if there are periodic points.
   - bifurcation : determination of the points of bifurcation. The user gives an interval of α the period and the number of periodic points at the beginning of this interval. The program finds the value of α where there is a new stable periodic orbit (point where there are more fixed points than in the beginning of the interval and where there is a stable periodic orbit). Between each iteration, the user can stop the search by typing 0.
   - chan_alpha : new parameters m_0, m_1, α, β.
   - erase : clears the graphic screen.

- initial : iterations of the function at a chosen point. The user chooses the point and the number of iterations, the program maps and gives the value of each iteration.
- intervals : determination of the invariant intervals of the 1-Dmap. The user chooses the interval of α (alphamin, alphamax) where the program will run, the value of β. Then he selects the type of the extremum (3 for minimum, 4 for maximum). Finally, the user chooses the number of the extremum (0, 1, 2 for the first, second or third maximum or minimum) : the program finds α when the image of the first maximum is equal to this extremum.
- kinvariant : addresses and dynamical representation of the extrema.
- periodic : determination of the periodic points for the parameter values chosen at the beginning of the program or after the option roam. The user simply selects the period n and the desired graphic output : 1-D map iterated n times or a picture of the usual 1-D map and the itinerary of each periodic point mapped on it. The figure also gives the slope at the periodic points and the total number of periodic points. If the slope is in the interval [-1, 1], the point is in a stable orbit.
- quit : end of the program.
- roam : new scale for the figure. The user gives the boundaries of the picture (xmin, xmax, ymin, ymax).
- separation : determination of the extrema of the 1-D map.

2. Algorithms

2.1 Determination of periodic points of period n

This algorithm is used by the options periodic, a-beta, bifurcation.

It finds the fixed points of f^n, for α, in the interval [0., 5.] (it stops at x = 5. because for $\beta = 15.$ there are no periodic points greater than 5.). The general form of this algorithm is :

- the program takes the first interval. Its length is equal to dx.
- it checks if the boundaries are fixed points.
  - if yes, it determines the slope at this point and restarts with the next interval of length dx.
  - if not, it determines the sign of $(f^n(x) - x)$ at the boundaries of the interval. If the sign is different there is a fixed point in this interval (the function $f^n(x) - x$ is continuous, if its sign changes in an interval there is a point where it is equal to 0). Therefore, in order not to miss a fixed point dx must be less than the smallest interval between two fixed points. In the other hand, if dx is too small, the program run time will be too long. We choose $dx = 10^{-5}$.
    - if the sign is equal, the algorithm continues with the next interval dx.
    - if the sign is different, it starts a more accurate search in this interval. It cuts the interval into two parts : one of length (0.618 dx) and the other (1 - 0.618)dx (with 0.618 the program reaches the search point faster). It calculates $(f^n(x) - x)$ at the boundaries of these two intervals. If these three points are not fixed points it determines in which interval the fixed point is (it is the interval whose boundaries have different signs for $(f^n(x) - x)$). This interval is cut again in the same way until the point of separation satisfies $(f^n(x) - x < 10^{-6})$. Then the program restarts with the next interval dx.

2.2 Find the extrema of the 1-D map

The option separation uses this algorithm.

In the following, we will give the algorithm of finding the minima, finding the maxima is similar (if we change the signs).

- the program takes one interval of length dx : [a , b]. It calculates f(b) - f(a) and f(b+dx) - f(b). If the first number is negative and the second positive, there is a minimum in one of these intervals.

- To find this minimum, the program cuts the first interval in two parts : [a, a'], [a', b].
  - If f(a') - f(a) is positive, the program restarts with the interval [a, a'].
  - If f(a') - f(a) is negative, the program restarts with the interval [a', b].
- the program stops if it finds the minimum ($f(a') - f(a) < 10^{-6}$) or if the minimum is not reached and the length of the interval is lower than 10^{-6} (in this case the minimum is not in the interval).

For the 1-D map there are four minima and four maxima.

2.3 Find the point of bifurcation

The option bifurcation uses this algorithm.

- For all values of α until the final value and for every α separated with a step equal to $(\alpha_{max}-\alpha_{min})/10$, the program calculates the 1-D map.
- Then it determines the periodic points and looks if there is a stable periodic orbit.
- If there are at least two more periodic points than in the previous iteration and if there is a stable periodic point, the actual value of α is a bifurcation point. In fact, there is a new periodic orbit of period n if there are n or 2n more points. When there are many periodic points the program does not find all of them (see determination of periodic points) but always finds at least two of them. For the first iteration the user must give the previous number of periodic points.
- If there are exactly two more unstable periodic points, the bifurcation point is between the previous iteration of α and the actual value of α. These values will be the new boundaries of the interval $[\alpha_{min}, \alpha_{max}]$. At this moment, the user can decide if the program should continue (by typing 1), or terminate (by typing 0).
- When the bifurcation point is reached the first boundaries of α are set again, the program increments the previous value of α, memorises the number of periodic points and begins to look for the next bifurcation point.

2.4 Research of the invariant intervals.

This algorithm is used in intervals

This program finds when the image of the first maximum is equal to the extremum number i (i and the type of the extremum (maximum or minimum) are chosen by the user).

- In an interval of α chosen by the user and for each value of α separated from the others by a step equal to one tenth of the length of the interval, the program determines the value of the maximum, its image, and the value of the extremum i.
  - if the image is greater, the step of α is divided by two, α is reset to its previous value and the program restarts from the beginning.
  - if the image is lower than the extremum the programs increments α. Before starting a new calculation, the program checks to see if it has already considered this α in the last two iterations (it can happen if the program had overshot the searched α in the preceding iteration) ; if yes, the step is divided by two before incrementating.

2.5 Kneading theory

This program gives the dynamical representation and the itineraries of the extrema.

For each extremum, the program calculates the images, determines the sequence $(v_n(i))_{n=0}^{\infty}$, then again calculates the images to obtain the itinerary.

3. Program

Acknowledgments

We would like to thank all the reviewers for many valuable comments and suggestions.

References

[1] L. O. Chua, C. A. Desoer and E. S. Kuh, *Linear and nonlinear circuits*, Mc Graw-Hill (1987).

[2] T. Matsumoto, L. O. Chua and M. Komuro, "The Double Scroll", *IEEE, Trans. CAS*, vol.CAS-32, pp 797-818 (August 1985).

[3] L. O. Chua, M. Komuro and T. Matsumuto, "The double scroll family", *IEEE Trans, Cas-31*, pp 1072-1118 (1986).

[4] L. Yang and Y. Liao, "Self-similar bifurcation structures from Chua's circuit", *International Journal of Circuit Theory and Application*, vol 15, pp 189-192 (1987).

[5] J. P. Keener, "The Sarkovskii sequence and stable periodic orbits of maps of the interval", *SIAM J. Numer. Anal.*, vol 23, No 5, pp 976-985 (October 1986).

[6] D. Singer, "Stable orbits and bifurcations of maps of the interval", *SIAM Jl. appl. Maths*, vol 35, pp 260-267 (1978).

[7] J. Guckenheimer, "Bifurcations of dynamical systems", in *Dynamical Systems*, CIME Lectures, Bressanone, Italy, June 1978. Boston, MA : Birkhauser, 1980, pp 115-231.

[8] J. Guckenheimer and P. J. Holmes, *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*, New York, Heidelberg, Berlin, Tokyo : Springler-Verlag (1983).

[9] P. J. Holmes and D. C. Whitley, "Bifurcations of one- and two- dimensional maps", *Phil. Trans. Roy. Soc. Lond.*, vol A311, pp 43-102 (1984).

[10] P. Collet and J.P. Eckmann, *Iterated maps on the interval as dynamical systems*, Boston : Birkhauser, 1980.

Figures

The pictures of the 1-D map are plotted for $m_0 = -0.14$ and $m_1 = 0.28$.

- fig 1 : Chua's circuit.
- fig 2 : symmetric nonlinearity (g).
- fig 3 : the Double Scroll for m_0=–0.5, m_1=–0.8, $B_p = 1$, $1/C_1 = 9$, $1/C_2 = 1$, $1/L = 7$ and $G = 0.7$. Projections onto the (i_L, v_{C_1})-plane, onto the (i_L, v_{C_2})-plane and onto the (v_{C_1}, v_{C_2})-plane.
- fig 4 : geometric structure of the Double Scroll.
- fig 5 : the 1-D map, a piecewise-continuous function ($\alpha = 10.$).
- fig 6 : behavior of the Double Scroll in (α β) space.
- fig 7 : 1-D map unimodal (α=9.1047).
- fig 8 : 1-D map multimodal (α=9.2).
- fig 9 : 1-D map iterated two times, first homoclinicity of period 2 (α=9.8).
- fig 10 : 1-D map, homoclinicity of period 1 (α=11.4).
- fig 11 : detail of figure 10.
- fig 12 : 1-D map iterated two times, second homoclinicity of period 2 (α=12.).
- fig 13 : 1-D map, death of periodic orbits (α=12.9).
- fig 14 : 1-D map iterated two times (α=9.337).
- fig 15 : 1-D map iterated three times (α=9.337).
- fig 16 : (a) periodic orbit of the Double Scroll when the 1-D map has one stable fixed point x_1 and when $0 < x_1 < x_0$. (b) simplification of the previous picture.
- fig 17 : periodic orbit of the Double Scroll when the 1-D map has one stable fixed point x_1 and when $x_1 > x_0$.
- fig 18 : periodic orbits of the Double Scroll System. (a) : period 2 for the Double Scroll and the 1-D map ($\alpha = 8.5$), (b) : period-5 orbit for the Double Scroll (period-3 for the 1-D map, $\alpha = 9.337$).

- **fig 19 : an unimodal map.**

```
/*************************************************************************
*************************************************************************
*************************************************************************/

#include <stdio.h>
#include <local/graf.h>

#ifndef MICROANGELO
#ifndef MASSCOMP
#define MASSCOMP
#endif
#endif

#define FALSE   0
#define TRUE    1
#define NIL(X)  ((X *) 0)

#define PROG_NAME       "ldmap"
#define BUF_SIZE        128

/*************************************************************************/

#ifdef MASSCOMP

#define MC_PLANES       5       /*offsets in hardware config array*/
#define MC_DISPLAY      6

#define MC_COLOR        1       /*values for DISPLAY in config array*/
#define MC_MONO         0

#define MC_CONFIG_SIZE  10

static mc_config[MC_CONFIG_SIZE];       /*hardware configuration array*/

#endif

/*************************************************************************/

#define HELP            0
#define CONTINUE        1
#define ERASE           2
#define MULTI_MAP       3
#define QUIT            4
#define INITIAL         5
#define CHAN_ALPHA      6
#define ROAM            7
#define POINCARE_MAP    8
#define ORIGINAL_MAP    9
#define BIFURCATION     10
#define SEPARATION      11
#define KINVARIANT      12
#define PERIODIC        13
#define A_BETA          14
#define INTERVALS       15


static char *opts_key[] =
{
    "?",
    "continue",
    "erase",
    "multi_map",
    "quit",
    "initial",
    "change alpha",
    "roam",
    "poincare_map",
    "original_map",
    "bifurcation",
    "separation",
    "kinvariant",
    "periodic",
    "a beta",
    "intervals",
    NIL(char)
};
/*************************************************************************/

/*error codes*/
#define GRAF_OPEN       1
#define UNRECOG         2
#define AMBIG           3

/*************************************************************************/

static int screen_drawn = FALSE; /*TRUE if screen drawn*/

static GRAF *graf;

#include <stdio.h>
#define TBL_SIZ 20480
#define STEP 0.01
#define EPS (1.e-8)

extern double sin(), cos(), exp(), fabs(), sqrt(), atan();
extern double get_real();
double *fx,*fy, *my, *py;
double s,t,m0,m1;
double got(),sot(),oot(),go(),so(),gpt(),spt(),opt(),gp(),sp();
double map_f(),search();
double ao,g0,a1,g1;
double ko,k1,k,qo,q1,po,p1;
double a11,a12,a21,a22;
double aox,aoy,box,boy;
double a1x,a1y,b1x,b1y;
double xmin,xmax,ymin,ymax;  /*  the scall of the screen  */
int eigen();
int p_flag, m_flag;



/*************************************************************************/
/*      creat the mapping list fxy[][] for ldmap                         */
/*************************************************************************/
map_1()
{
    double u,det;
    double atx,aty,atz,btx,bty,btz;
    double aoz, boz;
    double ub,vb;
    double yobx,yoby;
    double y1bx,y1by;
    double x1bx,x1by;
    double xb,yb;
    double xc,yc;
    double angle, abslute;
```

```
    int n, m, i;
    int size;

    /*  compute the normalized eigenvalue parameters */
    ao=so(s,t,m0);    /*  ao is sigam 0  */
    g0=go(s,t,m0);    /*  g0 is gama 0   */
    a1=sp(s,t,m1);    /*  a1 is sigam 1  */
    g1=gp(s,t,m1);    /*  g1 is gama 1   */
    k = - got(s,t,m0) / gpt(s,t,m1);

    ko=k;
    k1=1./k;

    qo=(ao-g0)*(ao-g0)+1.;      /*  (2.22)+  */
    q1=(a1-g1)*(a1-g1)+1.;      /*  (2.31)   */

    po=ao+ko*(ao*ao+1.)/g0;     /*  (2.21)   */
    p1=a1+k1*(a1*a1+1.)/g1;     /*  (2.27)   */

        printf("s,t,m0,m1=%f,%f,%f,%f\n",s,t,m0,m1);
        printf("ao,g0,a1,g1=%f,%f,%f,%f\n",ao,g0,a1,g1);

        /*   the L matrix in (4.44)   */
    det=(a1*a1+1.)*k1/((ao*ao+1.)*q1*(k+1.)*g1);
    a11=det*(-g1*(ko+1.)*(qo+g0*(ao-g0)*(k1+1.)));
    a12=det*(g0*g1*(k1+1.)*(ko+1.));
    a21 = - g0*(k1+1.)*(ao-g0)*(a1*(a1-g1)+1.);
    a21 += - g1*(ko+1.)*(a1-g1)*(ao*(ao-g0)+1.);
    a21*=det;
    a22=det*(g0*(k1+1.)*(q1+g1*(a1-g1)*(ko+1.)));

        /*    point A0    */
    aox=1.;
    aoy=po;
    aoz=0.;

        /*    point B0    */
    box=g0*(g0-ao-po)/qo;
    boy=g0*(1.-po*(ao-g0))/qo;
    boz=1.-box;

        /*    point A1    */
    a1x=1.;
    a1y=p1;

        /*    point B1    */
    b1x=1.;
    b1y=a1;

        size = TBL_SIZ / 2 ;

    xb=sqrt(b1x*b1x+b1y*b1y);
    abslute = xb;
    angle = atan(b1y);
    fx[0] = xb*exp(-a1*(3.14159+angle));
    fy[0] = xb*exp(a1*(3.14159-angle));

/*  spiral      BC      */
    u = 0;
    for ( n=1; n < size; n++ )
    {
        u = u + STEP;
        atx=exp(ao*u)*(aox*cos(u)-aoy*sin(u));
        aty=exp(ao*u)*(aox*sin(u)+aoy*cos(u));
        atz=exp(g0*u)*aoz;

        btx=exp(ao*u)*(box*cos(u)-boy*sin(u));
        bty=exp(ao*u)*(box*sin(u)+boy*cos(u));
        btz=exp(g0*u)*boz;
        ub=(btx+btz-1)/(btx+btz-atx-atz);
        vb=1-ub;

        yobx=ub*atx+vb*btx;
        yoby=ub*aty+vb*bty;

        y1bx=a11*(yobx-1)+a12*(yoby-po)+1;
        y1by=a21*(yobx-1)+a22*(yoby-po)+p1;

        x1bx=ub*a1x+vb*b1x;
        x1by=ub*a1y+vb*b1y;

/* ld map of BC                                          */

        xb=sqrt(x1bx*x1bx+x1by*x1by);
        xb*=exp(-a1*(3.14159+atan(x1by/x1bx)));
        yb=sqrt(y1bx*y1bx+y1by*y1by);
        if ( y1bx * y1bx > EPS )
        {
           if (y1by/y1bx >= b1y && y1bx > 0)
              yb *= exp(a1 * (3.14159 - atan(y1by/y1bx)));
           else if (y1bx < 0 && y1by >= 0)
              yb *= exp(a1 * atan ( -y1by / y1bx ));
           else if (y1bx > 0)
           {
              yb *= exp(a1 * (angle - atan(y1by/y1bx)));
              if (yb < abslute)
                 yb = fy[0] * yb / abslute;
              else
                 yb = fx[0] * yb / abslute;
           }
           else
           {
              yb *= exp( - a1 * atan(y1by/y1bx));
              if (yb < fx[0])
                 yb = fy[0] * yb / fx[0];
           }
        }
        else
        {
           if ( y1by > 0 )
              yb *= exp(a1 * 3.14159 * 0.5);
           else
              yb *= exp(a1 * 3.14159 * 1.5);
        }

        if (xb > fx[n-1])
        {
           fx[n] = xb;
           fy[n] = yb;
        }
        else
        {
           fx[n] = fx[n-1];
           fy[n] = fy[n-1];
        };
    };
/* spiral CA infinit                                     */
```

```
        u = STEP;
        atx=exp(ao*u)*(aox*cos(u)-aoy*sin(u));
        aty=exp(ao*u)*(aox*sin(u)+aoy*cos(u));
        atz=exp(g0*u)*aoz;

        btx=exp(ao*u)*(box*cos(u)-boy*sin(u));
        bty=exp(ao*u)*(box*sin(u)+boy*cos(u));
        btz=exp(g0*u)*boz;

        ub = (btx+btz+1)/(btx+btz-atx-atz);
        vb=1-ub;

        yobx= -ub*atx-vb*btx;
        yoby= -ub*aty-vb*bty;

        y1bx=a11*(yobx-1)+a12*(yoby-po)+1;
        y1by=a21*(yobx-1)+a22*(yoby-po)+p1;

        x1bx=ub*a1x+vb*b1x;
        x1by=ub*a1y+vb*b1y;

/* 1d map of A infinit    */

        xb=sqrt(x1bx*x1bx+x1by*x1by);
        fx[TBL_SIZ-1] = xb * exp(-a1*(3.14159+atan(x1by/x1bx)));
        yb=sqrt(y1bx*y1bx+y1by*y1by);
        if ( y1bx * y1bx > EPS )
        {
           if (y1by/y1bx >= b1y && y1bx > 0)
              yb *= exp(a1 * (3.14159 - atan(y1by/y1bx)));
           else if (y1bx < 0 && y1by >= 0)
              yb *= exp(a1 * atan ( -y1by / y1bx ));
           else if (y1bx > 0)
           {
              yb *= exp(a1 * (angle - atan(y1by/y1bx)));
              if (yb < abslute)
                 yb = fy[0] * yb / abslute;
              else
                 yb = fx[0] * yb / abslute;
           }
           else
           {
              yb *= exp( - a1 * atan(y1by/y1bx));
              if (yb < fx[0])
                 yb = fy[0] * yb / fx[0];
           }
        }
        else
        {
           if ( y1by > 0 )
              yb *= exp(a1 * 3.14159 * 0.5);
           else
              yb *= exp(a1 * 3.14159 * 1.5);
        }

        fy[TBL_SIZ-1] = yb;

    for ( n=1; n < size; n++ )
    {
        u = u + STEP;
        atx=exp(ao*u)*(aox*cos(u)-aoy*sin(u));
        aty=exp(ao*u)*(aox*sin(u)+aoy*cos(u));
        atz=exp(g0*u)*aoz;

        btx=exp(ao*u)*(box*cos(u)-boy*sin(u));
        bty=exp(ao*u)*(box*sin(u)+boy*cos(u));
        btz=exp(g0*u)*boz;

        ub = (btx+btz+1)/(btx+btz-atx-atz);
        vb=1-ub;

        yobx= -ub*atx-vb*btx;
        yoby= -ub*aty-vb*bty;

        y1bx=a11*(yobx-1)+a12*(yoby-po)+1;
        y1by=a21*(yobx-1)+a22*(yoby-po)+p1;

        x1bx=ub*a1x+vb*b1x;
        x1by=ub*a1y+vb*b1y;

/* 1d map of CA infinit    */

        xb=sqrt(x1bx*x1bx+x1by*x1by);
        xb = xb * exp(-a1*(3.14159+atan(x1by/x1bx)));
        yb=sqrt(y1bx*y1bx+y1by*y1by);
        if ( y1bx * y1bx > EPS )
        {
           if (y1by/y1bx >= b1y && y1bx > 0)
              yb *= exp(a1 * (3.14159 - atan(y1by/y1bx)));
           else if (y1bx < 0 && y1by >= 0)
              yb *= exp(a1 * atan ( -y1by / y1bx ));
           else if (y1bx > 0)
           {
              yb *= exp(a1 * (angle - atan(y1by/y1bx)));
              if (yb < abslute)
                 yb = fy[0] * yb / abslute;
              else
                 yb = fx[0] * yb / abslute;
           }
           else
           {
              yb *= exp( - a1 * atan(y1by/y1bx));
              if (yb < fx[0])
                 yb = fy[0] * yb / fx[0];
           }
        }
        else
        {
           if ( y1by > 0 )
              yb *= exp(a1 * 3.14159 * 0.5);
           else
              yb *= exp(a1 * 3.14159 * 1.5);
        }

        if (xb < fx[TBL_SIZ-n])
        {
           fx[TBL_SIZ-1-n] = xb;
           fy[TBL_SIZ-1-n] = yb;
        }
        else
        {
           fx[TBL_SIZ-1-n] = fx[TBL_SIZ-n];
           fy[TBL_SIZ-1-n] = fy[TBL_SIZ-n];
        };
    };
}

/*************************************************************************
```

```
    the ldmap function map_f(x)
*************************************************************************/
static double
map_f(x)
double x;
{

    if (x <= fx[0])
    /*   in the linear map region  */
       return(fy[0] * x / fx[0]) ;
    else
       return(search(x));
}

/*************************************************************************
     search for f(x)
*************************************************************************/
static double search(x)
double x;
{
    int min,max,mid;
    double y;

    min = 0;
    max = TBL_SIZ - 1;
    while (max > min + 1)
    {
       mid = ( min + max ) / 2;
       if ( x < fx[mid] )
          max = mid;
       else
          min = mid;
    };


    if ((fx[max]-fx[min])<=EPS)
       return(fy[min]);
    else
       return((x-fx[min])*(fy[max]-fy[min])/(fx[max]-fx[min]) + fy[min]);
}




/*************************************************************************/
/*************************************************************************/

static
gp_init()

{
#ifdef MICROANGELO
    cinit(0x70,4,0xf0,0xf2,0xf4,0xf6,0,0,0,0,3);
#else ifdef MASSCOMP
    mgiasngp(0, 0);             /*assign graphics library*/
    mgigethc(MC_CONFIG_SIZE, mc_config);
    if ((mc_config[MC_DISPLAY] == MC_MONO && mc_config[MC_PLANES] != 2) ||
        (mc_config[MC_DISPLAY] == MC_COLOR && mc_config[MC_PLANES] != 6))
    {
        fprintf(stderr, "Not proper graphics display\n");
        exit(1);
    }
#endif
    erase_screen();
}

/*************************************************************************/
/*************************************************************************/

static
draw_screen()

{
    graf_init();
    set_real(xmin, xmax, ymin, ymax, graf);
    set_x_axis(5, 5, 5, 4, 1, 1.0, "X", 1.0, graf);
    set_y_axis(5, 5, 5, 4, 1, 1.0, "Y", 1.0, graf);
    set_title("1 dim map", 1, 1.0, graf);
#ifdef MICROANGELO
    set_screen(50, 500, 50, 400, graf);
#else ifdef MASSCOMP
    {
        int xmin, ymin, xmax, ymax, placed;

        mgigetvcoor(2, &xmin, &ymin, &xmax, &ymax, &placed);
        set_screen(xmin + 75, xmax - 50, ymin + 75, ymax - 50, graf);
    }
#endif
    draw_bounds(1, 1, 1, 1, 1, graf);
    screen_drawn = TRUE;
    fprintf( stderr, "Hello\n" );
}

/*************************************************************************/
/*************************************************************************/

static
graf_init()

{
    static initialized = FALSE;
    extern GRAF *graf_open();

    if (!initialized)
    {
        gp_init();
        if ((graf = graf_open()) == NULL)
            err_hdlr(GRAF_OPEN);
        initialized = TRUE;
    }
}


/*************************************************************************/
/*************************************************************************/

static
erase_screen()

{
#ifdef MICROANGELO
    erase(1, 1);
#else ifdef MASSCOMP
    if (mc_config[MC_DISPLAY] == MC_MONO)       /*2-plane mono*/
    {
        mgipln(3);                      /*use text plane*/
        mgiclearpln(2, -1, 0);  /*clear all planes*/
```

```
        mgihue(3);                      /*white*/
    }
    else        /*six-plane color*/
    {
        mgiclearpln(2, 31, 0);  /*clear all but text*/
        mgipln(21);
        mgihue(21);
    }
#endif
    screen_drawn = FALSE;
}

/*************************************************************************
*************************************************************************/
/*************************************************************************/
/*  1-d map iterated n times - x */
/*************************************************************************/
static double
func(x,n)
double x;
int n;
{
double ppy;
int m;
ppy = x;
for(m=0;m<n;m++)
ppy = map_f(ppy);
return(ppy-x);
}
/*************************************************************************/
/*************************************************************************/
/* MAIN PROGRAM  */
/*************************************************************************/
/*************************************************************************/
#define MAX_AA 2.5
#define N_AA 250000.0
#define ERROR 1.0e-6
char    outfile[128];
int bound,go_on;
double dx = MAX_AA / N_AA, a0x0,b0y0,aaxo,bbxo,abx,xy;
double fabs();
FILE   *fp;


/*************************************************************************/
main(argc,argv)
char **argv;
int    argc;

{

    double x,y,alp,bet,amin,amax,bmin,bmax,amin0,amax0,bmin0,bmax0,alp0,bet0;
    double gam,sig,omg,aax,bbx,asp,bsp,asp0,bsp0,aax0,bbx0,xx,ppy;
    double xb,yb;
    double x0,slo;
    double maxi[6],mini[6];
    double maxfx;
    int flag,stable,prec;
    int n,m,nm,mm,pp,ntt,nn,nt,cont,ii,jj,i,j,k,dyy,aa;
    int ite;
    char *malloc(),prece;

/************************
    m0 = - 1.0 / 7.0;
    m1 = 2.0 / 7.0;
************************/
 if (argc ==1)
    fp = stdout;
 else if (argc > 2 || argc == 2 && (fp = fopen (argv[1],"w")) == NULL) {
    fprintf (stderr, "Usage: %s [outfile]\n",argv[0]);
    exit (1);
    }
    m_flag = 0;
    p_flag = 0;
    /*   allocate the table space   */
    if ( (fx = (double *) malloc( TBL_SIZ * sizeof( double ) )) == NULL )
        abort( "not enough memory space\n" );
    if ( (fy = (double *) malloc( TBL_SIZ * sizeof( double ) )) == NULL )
        abort( "not enough memory space\n" );
    if ( (my = (double *) malloc( TBL_SIZ * sizeof( double ) )) == NULL )
        abort( "not enough memory space\n" );
    if ( (py = (double *) malloc( TBL_SIZ * sizeof( double ) )) == NULL )
        abort( "not enough memory space\n" );

begin:
    do
    {
        fprintf( stderr, "m0 : ");
        m0 = get_real();
        fprintf( stderr, "m1 : ");
        m1 = get_real();
        fprintf( stderr, "alpha : ");
        s = get_real();
        fprintf( stderr, "beta : ");
        t = get_real();
    }while ( eigen(s,t,m0,&gam,&sig,&omg)==-1 ||
             eigen(s,t,m1,&gam,&sig,&omg)==-1);


    xmin = 0.;
    xmax = 5.;
    ymin = 0.;
    ymax = 5.;
    draw_screen();

    graf_move(0.,0.,graf);
    graf_draw(10.,10.,graf);
    map_1();

/*
    xb=sqrt(b1x*b1x+b1y*b1y);
    xb*=exp(-a1*(3.14159+atan(b1y/b1x)));
    yb=sqrt(b1x*b1x+b1y*b1y);
    yb*=exp(a1*(3.14159/2-atan(-b1x/b1y)));
    graf_move(0.,0.,graf);
    graf_draw(xb,yb,graf);

    yc=sqrt(c1x*c1x+c1y*c1y);
    yc*=exp(a1*(3.14159/2-atan(-c1x/c1y)));

    graf_move(0.,yc,graf);
    graf_draw(xc,yc,graf);
*/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  draw 1Dmap curve  */
    graf_move(0.,0.,graf);
    graf_draw(fx[0],fy[0],graf);
    for (n=0; n<TBL_SIZ; n++)
```

```
        graf_point(fx[n],fy[n],graf);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  birth boundary of double scrall  */
    x0=sqrt(alx*alx+aly*aly);
    x0 = x0 * exp(-al*(3.14159+atan(aly/alx)));
    graf_move(0.,x0,graf);
    graf_draw(x0,x0,graf);
    graf_draw(x0,0.,graf);
    fprintf(fp,"a,b =%f  %f \n",s,t);
    fprintf(fp,"x0 =%f\n",x0);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
     interactive(&flag);
     switch(flag)
     {
        case 1:
           break;
        case 2:
           draw_screen();
           graf_move(0.,0.,graf);
           graf_draw(10.,10.,graf);
           goto interaction;
           break;
        case 3:
           m_flag = 1;
           m_map();
           break;
        case 5:
           break;
        case 6:
           goto begin;
           break;
        case 7:
           fprintf( stderr, "xmin : ");
           xmin = get_real();
           fprintf( stderr, "xmax : ");
           xmax = get_real();
           fprintf( stderr, "ymin : ");
           ymin = get_real();
           fprintf( stderr, "ymax : ");
           ymax = get_real();
           printf("xmin,xmax,ymin,ymax=%f,%f,%f,%f\n",xmin,xmax,ymin,ymax);
           draw_screen();
           goto interaction;
           break;
        case 8:
           p_flag = 1;
           p_map();
           break;
        case 9:
           p_flag = 0;
           break;
        case 10:
           p_flag = 1;
           goto bif;
           break;
        case 11:
           p_flag = 1;
           goto separ;
           break;
        case 12:
           p_flag = 1;
           goto kneading;
           break;
        case 13:
           p_flag = 1;
           goto perio;
           break;
        case 14:
           p_flag = 1;
           goto albet;
           break;
        case 15:
           p_flag = 1;
           goto inter;
           break;
      };


/*************************************************************************/
/* iteration of 1-D map starting from a chosen point */
/*************************************************************************/
initial:
      maxfx = fx[TBL_SIZ-1];
    fprintf( stderr, "initial point x(0) : ");
    x = get_real();
    while (x > maxfx)
    {
       fprintf( stderr, "try a smaller initial point x(0) : ");
       x = get_real();
    };
    printf("x(0)=%f\n",x);
iteration:
    fprintf( stderr, "the number of iteration : ");
    ite = (int) get_real();
    printf("n=%d\n",ite);

    for (n=0; n<ite; n++)
    {
        if (x > maxfx)
        {
           fprintf( stderr, "go to the big limit cycle. ");
           break;
        };
        if ( p_flag == 0 )
           y = map_f(x);
        else
        {
           y = map_f(x);
           while (y < fx[0])
              y = map_f(y);
        }
        graf_move(x,x,graf);
        graf_draw(x,y,graf);
        graf_draw(y,y,graf);
        printf("   %15.10g\n",y);
        x = y;
     };
     goto interaction;
/*************************************************************************/
/*   periodic points in (alpha,beta)-space  */
/*************************************************************************/
albet:
/* reset screen */
mgiclearpln(2,-1,0);
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  boundaries of the intervals of alpha and beta */
```

```
printf("boundaries of alpha and beta %\n");
fprintf(stderr,"alphamin: ");
amin = get_real();
fprintf(stderr,"alphamax :");
amax = get_real();
fprintf(stderr,"bmin :");
bmin = get_real();
fprintf(stderr,"bmax : ");
bmax = get_real();
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* set graphic screen */
graf_init();
set_real(amin,amax,bmin,bmax,graf);
set_x_axis(6, 5, 5, 2, 1, 1.0, "A", 1.0, graf);
set_y_axis(6, 5, 5, 2, 1, 1.0, "B", 1.0, graf);
set_title("alpha-beta map", 1, 1.0, graf);
set_screen(75,600, 75, 500, graf);
draw_bounds(1, 1, 1, 1, 1, graf);
screen_drawn = TRUE;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
fprintf(stderr,"period : ");
pp = (int) get_real();
graf_move(0.,0.,graf);
graf_draw(0.,0.,graf);
asp = (amax - amin)/20.;
bsp = (bmax - bmin)/20.;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* research of periodic point for each alpha and beta */
for (alp =amin; alp <=amax; alp +=asp){
    fprintf(fp,"alpha %f\n",alp);
    for (bet =bmin; bet <=bmax; bet +=bsp){
        s = alp;
        t =bet;
        fprintf(fp,"b  %f\n",t);
        if (eigen(s,t,m0,&gam,&sig,&omg) != -1 && eigen(s,t,m1,&gam,&sig,&omg) != -1)
        {
         map_1();
         x0 = sqrt(alx*alx+aly*aly);
         x0 = x0*exp(-a1*(3.14159+atan(aly/alx)));
        for (aax0 =dx, bbx0 = aax0+dx, bound = TRUE, b0y0 = 0.0; aax0< MAX_AA; aax0 +=
        {
            aax = aax0;
            bbx = bbx0;
            a0x0 = b0y0;
            if (bound && (a0x0 = func(aax,pp)) == 0.0){
                for ( n = 1; n<pp; n++){
                    if(fabs(func(aax,n)) < ERROR)
                    goto fin1;
                }
                graf_point(alp,bet,graf);
                fprintf(fp,"fixed point %15.10g\n",aax);
fin1 :
                bound = TRUE;
            } else if ((b0y0 = func(bbx,pp)) == 0.0){
                for (n=1; n<pp; n++){
                    if (fabs(func(bbx,n)) < ERROR)
                    goto fin2;
                }
                graf_point(alp,bet,graf);
                fprintf(fp,"fixed point %15.10g\n",bbx);
fin2:
                aax0 +=dx;
                bbx0 +=dx;
                bound = TRUE;
            } else if (a0x0 > 0.0 && b0y0 < 0.0 || a0x0 < 0.0 && b0y0 > 0.0){
                bound = FALSE;
                for (abx = aax + (bbx - aax)* 0.618, go_on = TRUE;go_on; abx = aax+(bb
                    xy = func(abx,pp);
                        if (fabs(xy) < ERROR || fabs(aax - bbx) < ERROR){
                            for (n=1; n<pp; n++){
                                if(fabs(func(abx,n)) < ERROR)
                                   goto fin3;
                            }
                            graf_point(alp,bet,graf);
                            fprintf(fp,"fixed point : %15.10g\n",abx);
fin3:
                            go_on = FALSE;
                        }
                        else if (xy > 0.0)
                            if(a0x0 < 0.0)
                                bbx = abx;
                            else
                                aax = abx;
                        else if (xy<0.0)
                            if (a0x0 > 0.0)
                                bbx = abx;
                            else
                                aax = abx;
                        }
                } else
                    bound = FALSE;
            }
        }
    }
}
goto interaction;
/*********************************************************************/
/* research of bifurcation points (fixed beta)*/
/*********************************************************************/
bif:
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
fprintf(stderr,"alphamin: ");
amin = get_real();
fprintf(stderr,"alphamax :");
amax = get_real();
fprintf(stderr,"beta :");
bmin = get_real();
fprintf(stderr,"period : ");
pp = (int) get_real();
fprintf(stderr,"nber precedent orbits :");
ntt = (int) get_real();
nn = 0;
bfin:
asp = (amax - amin)/10.;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
for (alp =amin; alp <=amax; alp +=asp){
    fprintf(fp,"alpha %25.20g\n",alp);
    s = alp;
    t =bmin;
    nt = 0;
    stable = 0;
    if (eigen(s,t,m0,&gam,&sig,&omg) != -1 && eigen(s,t,m1,&gam,&sig,&omg) != -1)
    {
    map_1();
    x0 = sqrt(alx*alx+aly*aly);
    x0 = x0*exp(-a1*(3.14159+atan(aly/alx)));
    for (aax0 =dx, bbx0 = aax0+dx, bound = TRUE, b0y0 = 0.0; aax0< MAX_AA; aax0 +=dx, bbx
    aax = aax0;
```

```
    bbx = bbx0;
    a0x0 = b0y0;
    if (bound && (a0x0 = func(aax,pp)) == 0.0){
        for ( n = 1; n<pp; n++){
            if(fabs(func(aax,n)) < ERROR)
               goto bfin1;
        }
        slo = (func(aax+(dx/10.),pp)-func(aax-(dx/10.),pp))*(5./dx)+1.;
        /* the periodic point is stable */
        if(fabs(slo) < 1.)
            stable = 1;
        nt++;
bfin1 :
         bound = TRUE;
      } else if ((b0y0 = func(bbx,pp)) == 0.0){
          for (n=1; n<pp; n++){
              if (fabs(func(bbx,n)) < ERROR)
                 goto bfin2;
          }
          slo = (func(bbx+(dx/10.),pp)-func(bbx-(dx/10.),pp))*(5./dx)+1.;
          if(fabs(slo) < 1.)
              stable = 1;
          nt++;
bfin2:
          aax0 +=dx;
          bbx0 +=dx;
          bound = TRUE;
      } else if (a0x0 > 0.0 && b0y0 < 0.0 || a0x0 < 0.0 && b0y0 > 0.0){
          bound = FALSE;
          for (abx = aax + (bbx - aax)* 0.618, go_on = TRUE;go_on; abx = aax+(bbx-aax)*0.
               xy = func(abx,pp);
               if (fabs(xy) < ERROR || fabs(aax - bbx) < ERROR){
                   for (n=1; n<pp; n++){
                        if(fabs(func(abx,n)) < ERROR)
                             goto bfin3;
                   }
                   slo = (func(abx+(dx/100.),pp)-func(abx-(dx/100.),pp))*(50./dx)+1.;
                   if (fabs(slo) < 10.){
                        printf("slope : %15.10g\n",slo);
                        printf("fixed point : %15.10g\n",abx);
                    }
                    if(fabs(slo) < 1.)
                        stable = 1;
                    nt++;
bfin3:
                    go_on = FALSE;
               }
               else if (xy > 0.0)
                   if(a0x0 < 0.0)
                       bbx = abx;
                   else
                       aax = abx;
               else if (xy<0.0)
                   if (a0x0 > 0.0)
                        bbx = abx;
                    else
                        aax = abx;
          }
      } else
          bound = FALSE;
    }
 }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Are there new orbits ? */
bfin4:
   fprintf(fp,"total %d\n",nt);
   if(nt>(ntt + 1)){
   /* there are more orbits than for the previous value of alpha */
        if( stable != 1){
        /* the new orbit is unstable -> new more precise research*/
           if(nn == 0){
           /*there was no previous more precise research, storage of the point*/
                amin0 = amin;
                amax0 = amax;
                bmin0 = bmin;
                asp0 = asp;
                alp0 = alp;
                nn = 1;
           }
           /* change of the slope and the program will run again */
           amin = alp - asp;
           amax = alp;
           goto bfin;
        }
   if (stable == 1){
        printf("birth of periodic orbits%\n");
        printf("a,b %15.10g, %15.10g\n",alp,bmin);
        ntt = nt;
            if (nn == 1)
            {
            /* restitution of the point before the precise research */
                asp = asp0;
                amin = amin0;
                amax = amax0;
                bmin = bmin0;
                alp = alp0-asp;
                nn = 0;
            }
     }
   }
   /* New incrementation ? */
   fprintf(stderr,"continuation 0-1\n");
   cont = (int) get_real();
   if (cont == 0)
   goto interaction;
   }
   goto interaction;
/*************************************************************************/
/*    find periodic points                                               */
/*************************************************************************/
perio:
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* choice of the parameters  */
nt = 0;
fprintf(stderr,"period : ");
pp = (int) get_real();
printf("options :%\n");
fprintf(stderr, "iterated map :1");
printf(" %\n");
fprintf(stderr, "iterations :2");
printf(" %\n");
aa = (int) get_real();
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* picture of the map iterated pp times */
if (aa <= 1.1)
{
   erase_screen();
   draw_screen();
```

```
    graf_move(0.,0.,graf);
    graf_draw(10.,10.,graf);
            /***/
    x0 = sqrt(alx*alx+aly*aly);
    x0 = x0 * exp(-al * (3.14159+atan(aly/alx)));
    graf_move(0.,x0,graf);
    graf_draw(x0,x0,graf);
    graf_draw(x0,0.,graf);
            /***/
    graf_move(0.,0.,graf);
    x = 0.;
    y = 0.;
/* map from x = 0. to 5. step 0.0001 */
    for (m = 0; m < 50000 ; m++)
    {
        for (n=0; n<pp ; n++)
            y = map_f(y);
            graf_draw(x,y,graf);
            x = x + 0.0001;
            y = x;
        }
    }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*   research of periodic points */
for (aax0 =dx, bbx0 = aax0+dx, bound = TRUE, b0y0 = 0.0; aax0< MAX_AA; aax0 +=dx, bbx0 +=
    aax = aax0;
    bbx = bbx0;
    a0x0 = b0y0;
    /* if aax is a periodic point */
    if (bound && (a0x0 = func(aax,pp)) == 0.0){
        /*elimination of the periodic points of a lower period */
        for ( n = 1; n<pp; n++){
            if(fabs(func(aax,n)) < ERROR)
            goto pfin1;
        }
         /* slope at the periodic point */
         slo = (func(aax+(dx/10.),pp)-func(aax-(dx/10.),pp))*(5./dx)+1.;
         fprintf(fp, "fixed-point %15.10g\n",aax);
         fprintf(fp ,"slope  %15.10g\n",slo);
         nt++;
pfin1 :
         bound = TRUE;
     } else if ((b0y0 = func(bbx,pp)) == 0.0){
         /* bbx is a periodic point */
         for (n=1; n<pp; n++){
             if (fabs(func(bbx,n)) < ERROR)
             goto pfin2;
         }
         slo = (func(bbx+(dx/10.),pp)-func(bbx-(dx/10.),pp))*(5./dx)+1.;
         fprintf(fp, "fixed-point %15.10g\n",bbx);
         fprintf(fp ,"slope  %15.10g\n",slo);
         nt++;
pfin2:
         aax0 +=dx;
         bbx0 +=dx;
         bound = TRUE;
    } else if (a0x0 > 0.0 && b0y0 < 0.0 || a0x0 < 0.0 && b0y0 > 0.0){
    /* if the sign of f(x)-x changes */
         bound = FALSE;
         for (abx = aax + (bbx - aax)* 0.618, go_on = TRUE;go_on; abx = aax+(bbx-aax)*0.6
         /* research of the periodic point in ths interval */
             xy = func(abx,pp);
             if (fabs(xy) < ERROR || fabs(aax - bbx) < ERROR){
             /* f(abx) - abx < error -> abx is fixed point */
                 for (n=1; n<pp; n++){
                     if(fabs(func(abx,n)) < ERROR)
                     goto pfin3;
                     /* elimination of the periodic points of a lower period*/
                 }
                 slo = (func(abx+(dx/100.),pp)-func(abx-(dx/100.),pp))*(50./dx)+1.;
                 fprintf(fp,"fixed point : %15.10g\n",abx);
                 fprintf(fp,"slope : %15.10g\n",slo);
             /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * /
                 /* picture of the itinerary of the periodic point */
                 xx = abx;
                 if (aa >= 1.2){
                     for (mm=0; mm<pp ; mm++)
                     {
                         ppy = map_f(xx);
                         graf_move(xx,xx,graf);
                         graf_draw(xx,ppy,graf);
                         graf_draw(ppy,ppy,graf);
                         xx = ppy;
                     }
                 }
                 nt++;
pfin3:
                 go_on = FALSE;
                 }
                 else if (xy > 0.0)
                      if(a0x0 < 0.0)
                          bbx = abx;
                       else
                           aax = abx;
                  else if (xy<0.0)
                       if (a0x0 > 0.0)
                           bbx = abx;
                       else
                           aax = abx;
             }
         } else
             bound = FALSE;
    }
fprintf(fp,"total %d\n",nt);
dyy = 0;
goto interaction;

/*********************************************************************/
/*   research of the extrema of 1-D map  */
/*********************************************************************/
separ:
/* dyy = 2 if the user wants also the kneading invariants */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* dyy = 4 if the user wants  to determine the invariant intervals*/
if (dyy == 4)
goto slfin;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* research of the minima  */
for (aax0 = dx,bbx0 = aax0+dx, b0y0 = 0.0; aax0 < MAX_AA; aax0 +=dx, bbx0 +=dx){
    aax = aax0;
    bbx = bbx0;
    a0x0 = map_f(aax + dx) - map_f(aax);
    b0y0 = map_f(bbx + dx) - map_f(bbx);
    if (a0x0 < 0.0 && b0y0 > 0.0){
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
        /* the sign of f(x+dx)-f(x) changes between the two boundaries */
        /* -> there is a minimum between these two points */
        for (abx = aax+(bbx-aax)*0.618, go_on = TRUE; go_on; abx = aax +(bbx-aax)*0.618){
```

```
            xy = map_f(abx) - map_f(aax);
                if (fabs(xy) < ERROR){
                /*  abx is a minimum */
                    if (dyy != 2)
                        printf("minimum %15.10g\n",abx);
                    mini[ii] = abx;
                    /* end of research min of the invariant intervals*/
                    if (dyy == 3)
                    if (ii == nm)
                        goto s1fin;
                    ii++;
                    go_on = FALSE;
                }
                else if (xy > 0.0)
                    bbx = abx;
                else if (xy < 0.0)
                    aax = abx;
                else if (fabs(aax - bbx) < ERROR)
                    go_on = FALSE;
        }
    }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* research of the maxima  */
s1fin:
for (aax0 = dx,bbx0 = aax0+dx, b0y0 = 0.0; aax0 < MAX_AA; aax0 +=dx, bbx0 +=dx){
    aax = aax0;
    bbx = bbx0;
    a0x0 = map_f(aax + dx) - map_f(aax);
    b0y0 = map_f(bbx + dx) - map_f(bbx);
    if (a0x0 > 0.0 && b0y0 < 0.0){
        for (abx = aax+(bbx-aax)*0.618, go_on = TRUE; go_on; abx = aax +(bbx-aax)*0.618){
           xy = map_f(abx) - map_f(aax);
           if (fabs(xy) < ERROR){
                if (dyy !=2)
                    printf("maximum %15.10g\n",abx);
                maxi[jj] = abx;
                /* for the research of f(maxi[0]) = min we just need */
                /* the first maximum */
                if ( dyy == 3)
                    goto inter1;
                /* end of research of maxima for "invariant intervals" */
                if (dyy == 4)
                    if(jj == nm)
                        goto inter2;
                jj++;
                go_on = FALSE;
           }
           else if (xy < 0.0)
               bbx = abx;
           else if (xy > 0.0)
               aax = abx;
           else if (fabs(aax - bbx) < ERROR/10.)
               go_on = FALSE;
        }
    }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* kneading invariants - itineraries  */
s2fin:
if (dyy == 2)
{
    for (i = 0; i<ii; i++){
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* kneading invariant of maxima */
        fprintf(fp,"maxi %15.10g\n",maxi[i]);
        y = (maxi[i] - 0.00000000001);
        prece = '+';
          for(j = 0; j<30; j++){
             if (y<maxi[0] || y>mini[ii-1]){
             if (prece == '+')
             {
               fprintf(fp,"+");
               prece = '+';
               }
              else
              {
               fprintf(fp,"-");
               prece = '-';
              }
              }
              else
                for (k=0; k<ii-1; k++){
                  if(y<maxi[k+1] && y>mini[k])
                  {
                     if (prece == '+')
                     {
                     fprintf(fp,"+");
                     prece = '+';
                     }
                     else
                     {
                     fprintf(fp,"-");
                     prece = '-';
                     }
                  goto k1fin;
                  }
                  if(y>maxi[k] && y<mini[k]){
                     if (prece == '+')
                     {
                     fprintf(fp,"-");
                     prece = '-';
                     }
                     else
                     {
                     fprintf(fp,"+");
                     prece = '+';
                     }
                  goto k1fin;
                  }
                }
                if(y>maxi[ii-1] && y<mini[ii-1])
                     if (prece == '+')
                     {
                     fprintf(fp,"-");
                     prece = '-';
                     }
                     else
                     {
                     fprintf(fp,"+");
                     prece = '+';
                     }
                k1fin:
                y = map_f(y);
                }
                fprintf(fp," %\n");
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* itinerary of maxima */
```

```
        y = (maxi[i] - 0.00000000001);
          for(j = 0; j<30; j++){
             if (y<maxi[0]){
               fprintf(fp,"1");
               goto k2fin;
               }
               else
                 for (k=0; k<ii-1; k++){
                   if(y>maxi[k] && y<mini[k])
                   {
                   fprintf(fp,"%d",(2*k)+2);
                   goto k2fin;
                   }
                   if(y<maxi[k+1] && y>mini[k]){
                   fprintf(fp,"%d",(2*k)+3);
                   goto k2fin;
                   }
                 }
                 if(y>maxi[ii-1] && y<mini[ii-1])
                 fprintf(fp,"%d",2*ii);
                 if(y>mini[ii-1])
                 fprintf(fp,"%d",(2*ii) +1);
                 k2fin:
                 y = map_f(y);
                 }
                 fprintf(fp," %\n");
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* kneading invariant of minima */
        fprintf(fp,"mini %15.10g\n",mini[i]);
        y = (mini[i] + 0.00000000001);
        prece = '+';
          for(j = 0; j<30; j++){
             if (y<maxi[0] || y>mini[ii-1]){
                    if (prece == '+')
                    {
                    fprintf(fp,"+");
                    prece = '+';
                    }
                    else
                    {
                    fprintf(fp,"-");
                    prece = '-';
                    }
            }
            else
              for (k=0; k<ii-1; k++){
                if(y<maxi[k+1] && y>mini[k])
                {
                   if (prece == '+')
                   {
                   fprintf(fp,"+");
                   prece = '+';
                   }
                   else
                   {
                   fprintf(fp,"-");
                   prece = '-';
                   }
                goto kfin;
                }
                if(y>maxi[k] && y<mini[k]){
                   if (prece == '+')
                   {
                   fprintf(fp,"-");
                   prece = '-';
                   }
                   else
                   {
                   fprintf(fp,"+");
                   prece = '+';
                   }
                goto kfin;
                }
              }
              if(y>maxi[ii-1] && y<mini[ii-1])
                   if (prece == '+')
                   {
                   fprintf(fp,"-");
                   prece = '-';
                   }
                   else
                   {
                   fprintf(fp,"+");
                   prece = '+';
                   }
              kfin:
              y = map_f(y);
              }
              fprintf(fp,"%\n");
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
        y = (mini[i] + 0.00000000001);
          for(j = 0; j<30; j++){
             if (y<maxi[0]){
               fprintf(fp,"1");
               goto k3fin;
               }
               else
                 for (k=0; k<ii-1; k++){
                   if(y>maxi[k] && y<mini[k])
                   {
                   fprintf(fp,"%d",2*k+2);
                   goto k3fin;
                   }
                   if(y<maxi[k+1] && y>mini[k]){
                   fprintf(fp,"%d",2*k+3);
                   goto k3fin;
                   }
                 }
                 if(y>maxi[ii-1] && y<mini[ii-1])
                 fprintf(fp,"%d",2*ii);
                 if(y>mini[ii-1])
                 fprintf(fp,"%d",(2*ii) +1);
                 k3fin:
                 y = map_f(y);
                 }
                 fprintf(fp," %\n");
                 }
                 }
                 dyy = 0;
                 ii = 0;
                 jj = 0;
    goto interaction;
/*********************************************************************/
/*  kneading theory */
/*********************************************************************/
    kneading:
    dyy = 2;
    ii = 0;
```

```
    jj = 0;
    goto separ;
/*************************************************************************/
/* invariant intervals  */
/*************************************************************************/
/* this program finds alpha when f(max 0)=f(max i or min i)  */
inter:
printf(" boundaries of alpha %\n");
fprintf(stderr,"alphamin: ");
amin = get_real();
fprintf(stderr,"alphamax :");
amax = get_real();
fprintf(stderr,"beta :");
bet = get_real();
fprintf(stderr,"type 3 for research of the min, 4 for the max : ");
dyy = (int) get_real();
fprintf(stderr,"# of minimum or maximum :");
nm = (int) get_real();
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
if (dyy == 3){
    asp = (amax - amin)/10.;
    for (alp =amin, go_on = TRUE; go_on; alp +=asp){
        fprintf(fp,"alpha %25.20g\n",alp);
        s = alp;
        t =bet;
        if (eigen(s,t,m0,&gam,&sig,&omg) != -1 && eigen(s,t,m1,&gam,&sig,&omg) != -1)
        {
            map_1();
            x0 = sqrt(alx*alx+aly*aly);
            x0 = x0*exp(-al*(3.14159+atan(aly/alx)));
            ii = 0;
            jj = 0;
            goto separ;
inter1:
            go_on = TRUE;
            fprintf(fp,"f %15.10g\n",map_f(maxi[0]));
            if (fabs(map_f(maxi[0]) - mini [nm]) < ERROR*10.){
                fprintf(fp,"alpha : %25.20g\n",alp);
                go_on = FALSE;
            }
            if (map_f(maxi[0]) > mini[nm]){
                prec = alp;
                alp = alp - asp;
                asp = asp * 0.5;
                go_on = TRUE;
            }
            else
                if((alp+asp) == prec)
                    asp = asp * 0.5;
        }
    }
}
if (dyy == 4){
    asp = (amax - amin)/10.;
    for (alp =amin, go_on = TRUE; go_on; alp +=asp){
        fprintf(fp,"alpha %25.20g\n",alp);
        s = alp;
        t =bet;
        if (eigen(s,t,m0,&gam,&sig,&omg) != -1 && eigen(s,t,m1,&gam,&sig,&omg) != -1)
        {
            map_1();
            x0 = sqrt(alx*alx+aly*aly);
            x0 = x0*exp(-al*(3.14159+atan(aly/alx)));
            ii = 0;
            jj = 0;
            goto separ;
inter2:
            go_on = TRUE;
            fprintf(fp,"f %15.10g\n",map_f(maxi[0]));
            if (fabs(map_f(maxi[0]) - maxi [nm]) < ERROR*10.){
                fprintf(fp,"alpha : %25.20g\n",alp);
                go_on = FALSE;
            }
            if (map_f(maxi[0]) > maxi[nm]){
                prec = alp;
                alp = alp - asp;
                asp = asp * 0.5;
                go_on = TRUE;
            }
            else
                if((alp+asp) == prec)
                    asp = asp * 0.5;
        }
    }
}

interaction:
    interactive(&flag);
    switch(flag)
    {
       case 1:
          goto iteration;
          break;
       case 2:
          draw_screen();
          graf_move(0.,0.,graf);
          graf_draw(10.,10.,graf);
          goto interaction;
          break;
       case 3:
          m_flag = 1;
          m_map();
          goto interaction;
          break;
       case 5:
          goto initial;
          break;
       case 6:
          goto begin;
          break;
       case 7:
          fprintf( stderr, "xmin : ");
          xmin = get_real();
          fprintf( stderr, "xmax : ");
          xmax = get_real();
          fprintf( stderr, "ymin : ");
          ymin = get_real();
          fprintf( stderr, "ymax : ");
          ymax = get_real();
          /*ymin = xmin;
          ymax = xmax;*/
          printf("xmin,xmax,ymin,ymax=%f,%f,%f,%f\n",xmin,xmax,ymin,ymax);
          draw_screen();
          goto interaction;
          break;
       case 8:
          p_flag = 1;
          p_map();
```

```
            goto interaction;
            break;
         case 9:
            p_flag = 0;
            /*  draw 1Dmap curve  */
            graf_move(0.,0.,graf);
            graf_draw(fx[0],fy[0],graf);
            for (n=0; n<TBL_SIZ; n++)
               graf_point(fx[n],fy[n],graf);
            goto interaction;
            break;
         case 10:
            p_flag = 1;
            goto bif;
            break;
         case 11:
            p_flag = 1;
            goto separ;
            break;
         case 12:
            p_flag = 1;
            goto kneading;
            break;
         case 13:
            p_flag = 1;
            goto perio;
            break;
         case 14:
            p_flag = 1;
            goto albet;
            break;
         case 15:
            p_flag = 1;
            goto inter;
            break;
      };
}

/*****************************************************************************
******************************************************************************/
m_map()
{
   int i,j, n;
   double x, y;

   do
   {
      fprintf( stderr, "multiplity : ");
      m_flag = get_int();
   }  while (m_flag < 1 );

   for (i=0; i<TBL_SIZ; i++)
   {
     x = fx[i];
     for (j=0; j<m_flag; j++)
        x = map_f(x);
     my[i] = x;
   }

/*  draw 1Dmap curve  */
    for (n=0; n<TBL_SIZ; n++)
       graf_point(fx[n],my[n],graf);
}
/*****************************************************************************
******************************************************************************/
p_map()
{
   int i,j,n;
   double x;

   for (i=0; i<TBL_SIZ; i++)
   {
     x = fx[i];
     x = map_f(x);
     while (x < fx[0])
        x = map_f(x);
     py[i] = x;
   }

/*  draw 1Dmap curve  */
    graf_move(0.,0.,graf);
    graf_draw(fx[0],py[0],graf);
    for (n=0; n<TBL_SIZ; n++)
       graf_point(fx[n],py[n],graf);
}

/*****************************************************************************
******************************************************************************/
static
eigen(s,t,m,gam,sig,omg)

double s,t,m,*gam,*sig,*omg;

{
    int i;
    double a,b,c,h,x,f,df;

    a=s*m+1.;
    b=s*(m-1.)+t;
    c=s*m*t;
    h=.5;
    x=1.;

    i=0;
    for (;;)
    {
        f=x*x*x+a*x*x+b*x+c;
        if ( fabs(f) < EPS )
        {
            *gam=x;
            *sig = -.5*(a+x);
            if ( (a+x)*(a+x)+4.*c/x > 0. )
            {
                printf("no complex\n");
                return(-1);
            }
            *omg=.5*sqrt(-(a+x)*(a+x)-4.*c/x);
            break;
        }
        df=3.*x*x+2.*a*x+b;
        if ( df < EPS )
        {
            printf("df singular\n");
            return(-1);
        }
        x-=h*f/df;
        i+=1;
        if ( i > 5000 )
```

```
        {
            printf("eigen failed\n");
            return(-1);
        }
    }
}

/*****************************************************************************
*****************************************************************************/
static double
got(s,t,m0)

double s,t,m0;

{
    double gam,sig,omg;

    eigen(s,t,m0,&gam,&sig,&omg);
    return(gam);
}

/*****************************************************************************
*****************************************************************************/
static double
sot(s,t,m0)

double s,t,m0;

{

    double gam,sig,omg;

    eigen(s,t,m0,&gam,&sig,&omg);
    return(sig);
}

/*****************************************************************************
*****************************************************************************/
static double
oot(s,t,m0)

double s,t,m0;

{
    double gam,sig,omg;

    eigen(s,t,m0,&gam,&sig,&omg);
    return(omg);

}

/*****************************************************************************
*****************************************************************************/
static double
go(s,t,m0)

double s,t,m0;

{
    double got(),oot();

    return(got(s,t,m0)/oot(s,t,m0));
}
```

```
/*****************************************************************************
*****************************************************************************/
static double
so(s,t,m0)

double s,t,m0;

{
    double sot(),oot();

    return(sot(s,t,m0)/oot(s,t,m0));
}

/*****************************************************************************
*****************************************************************************/
static double
gpt(s,t,m1)

double s,t,m1;

{
    double gam,sig,omg;

    eigen(s,t,m1,&gam,&sig,&omg);
    return(gam);
}

/*****************************************************************************
*****************************************************************************/
static double
spt(s,t,m1)

double s,t,m1;

{
    double gam,sig,omg;

    eigen(s,t,m1,&gam,&sig,&omg);
    return(sig);
}

/*****************************************************************************
*****************************************************************************/
static double
opt(s,t,m1)

double s,t,m1;

{
    double gam,sig,omg;

    eigen(s,t,m1,&gam,&sig,&omg);
    return(omg);
}

/*****************************************************************************
*****************************************************************************/
static double
gp(s,t,m1)

double s,t,m1;

{
    double gpt(),opt();
```

```
    return(gpt(s,t,ml)/opt(s,t,ml));
}

/*****************************************************************************
*****************************************************************************/
static double
sp(s,t,ml)

double s,t,ml;

{
    double spt(),opt();

    return(spt(s,t,ml)/opt(s,t,ml));
}

/*****************************************************************************
*****************************************************************************/
static
interactive(flag)
int *flag;

{
    char buf[BUF_SIZE];
    int i;

    while (TRUE)
    {
        fprintf(stderr, "\nEnter an option (%s for help): ",
            opts_key[HELP]);
        if (fgets (buf, BUF_SIZE, stdin) == NIL(char))
            return;
        buf[strlen(buf)-1] = '\0';      /*strip newline*/
        if (strlen(buf) == 0)   /*empty input line*/
        {
            for (i = 0; i < 64; i++)    /*clear text screen*/
                fprintf(stderr, "\n");
        }
        else
        {
            switch (key_match(buf, opts_key))
            {
            case CONTINUE:
                *flag = 1;
                return;
            case INITIAL:
                *flag = 5;
                return;
            case A_BETA:
                *flag = 14;
                return;
            case BIFURCATION:
                *flag = 10;
                return;
            case INTERVALS:
                *flag = 15;
                return;
            case PERIODIC:
                *flag = 13;
                return;
            case SEPARATION:
                *flag = 11;
                return;
            case KINVARIANT:
                *flag = 12;
                return;
            case CHAN_ALPHA:
                *flag = 6;
                return;
            case ROAM:
                *flag = 7;
                return;
            case MULTI_MAP:
                *flag = 3;
                return;
            case POINCARE_MAP:
                *flag = 8;
                return;
            case  ORIGINAL_MAP:
                *flag = 9;
                return;
            case ERASE:
                erase_screen();
                *flag = 2;
                return;
            case HELP:
                do_help();
                *flag = 1;
                break;
            case QUIT:
                free( fx );
                free( fy );
                free( my );
                free( py );
                exit(0);
            case -1:
                err_hdlr(UNRECOG);
                break;
            case -2:
                err_hdlr(AMBIG);
                break;
            }
        }
    }
}

/*****************************************************************/
/*****************************************************************/


static
do_help()

{
    fprintf(stderr, "\nOptions:\n");
    key_print(stderr, "    %s\n", opts_key);
    fprintf(stderr, "\n");
}

/*****************************************************************************/
/*****************************************************************************/

static
pr_fund_pts()

{
    fprintf( stderr, "alpha,beta,m0,m1=%f,%f,%f,%f\n",s,t,m0,m1);
```

```
    fprintf( stderr, "ao,g0,a1,g1=%f,%f,%f,%f\n",ao,g0,a1,g1);
    fprintf( stderr, "ko,k1,qo,q1,po,p1=%f,%f,%f,%f,%f,%f\n",ko,k1,qo,q1,po,p1);
    fprintf( stderr, "a11,a12,a21,a22=%f,%f,%f,%f\n",a11,a12,a21,a22);
    fprintf( stderr, "aox,aoy,box,boy=%f,%f,%f,%f\n",aox,aoy,box,boy);
    fprintf( stderr, "a1x,a1y,b1x,b1y=%f,%f,%f,%f\n",a1x,a1y,b1x,b1y);
}

/*****************************************************************/
/*****************************************************************/

static
err_hdlr(code)

int code;

{
    switch (code)
    {
    case GRAF_OPEN:
        fprintf(stderr, "%s: cannot open graflib\n", PROG_NAME);
        exit(1);
    case UNRECOG:
        fprintf(stderr, "unrecognizable command\n");
        break;
    case AMBIG:
        fprintf(stderr, "ambiguous command\n");
        break;
    default:
        fprintf(stderr, "illegal error code\n");
        exit(1);
    }
}

/*****************************************************************/
```

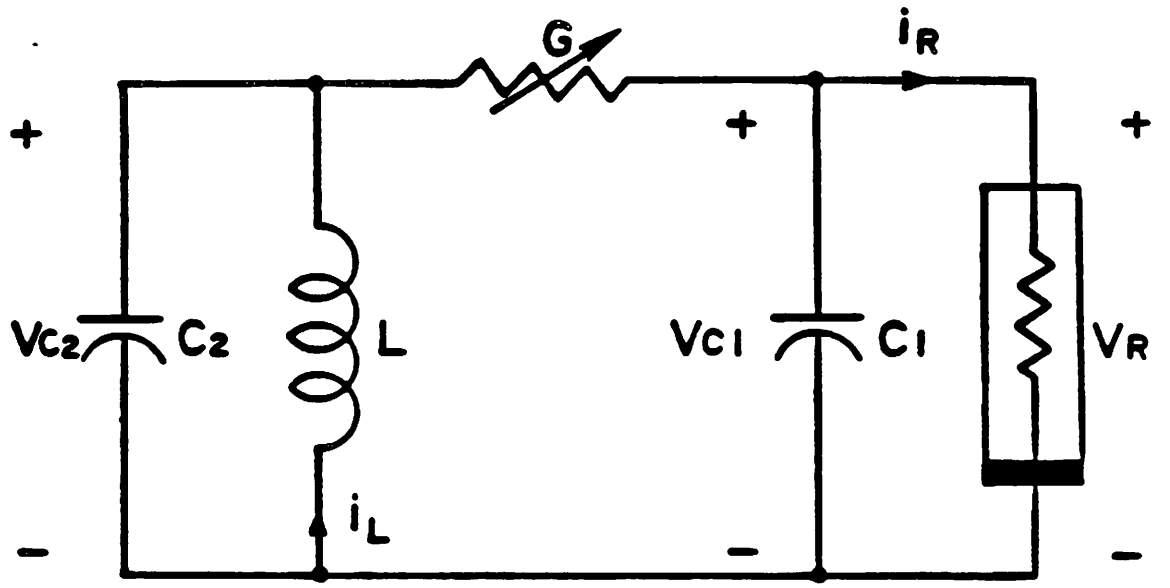

Fig. 1

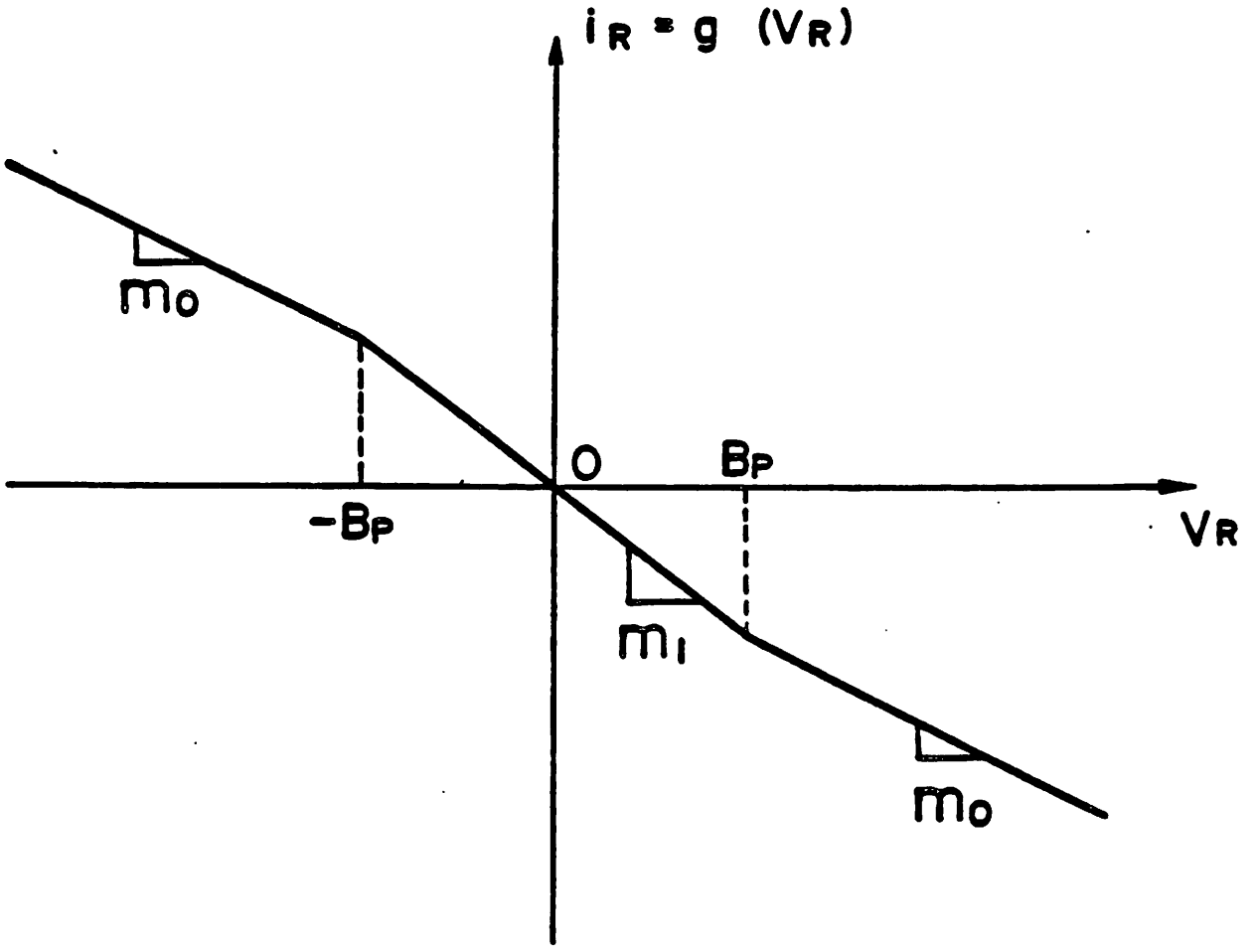

Fig. 2

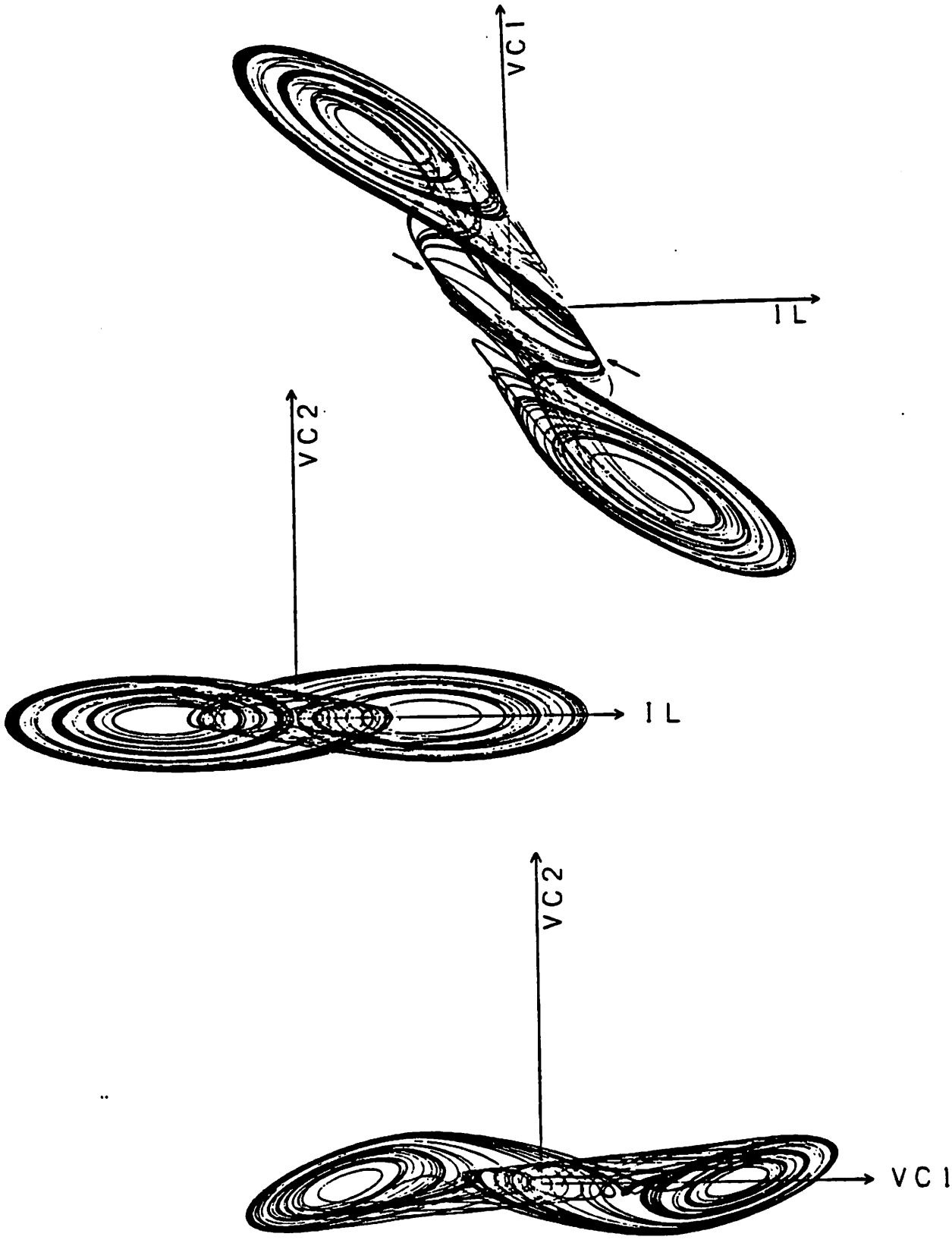

Fig. 3

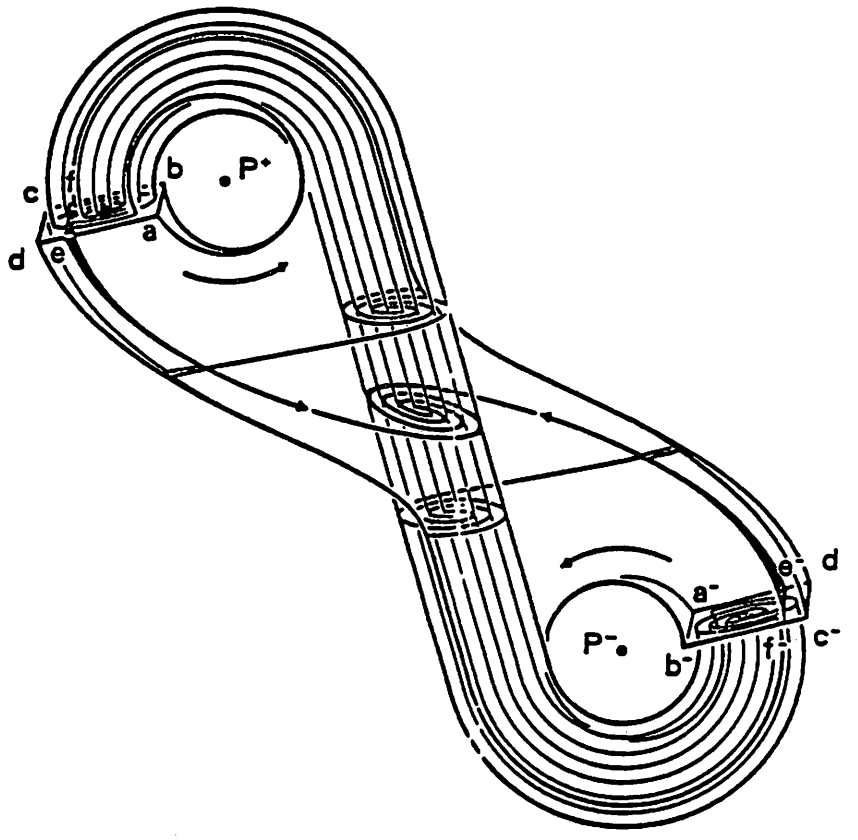

Fig. 4

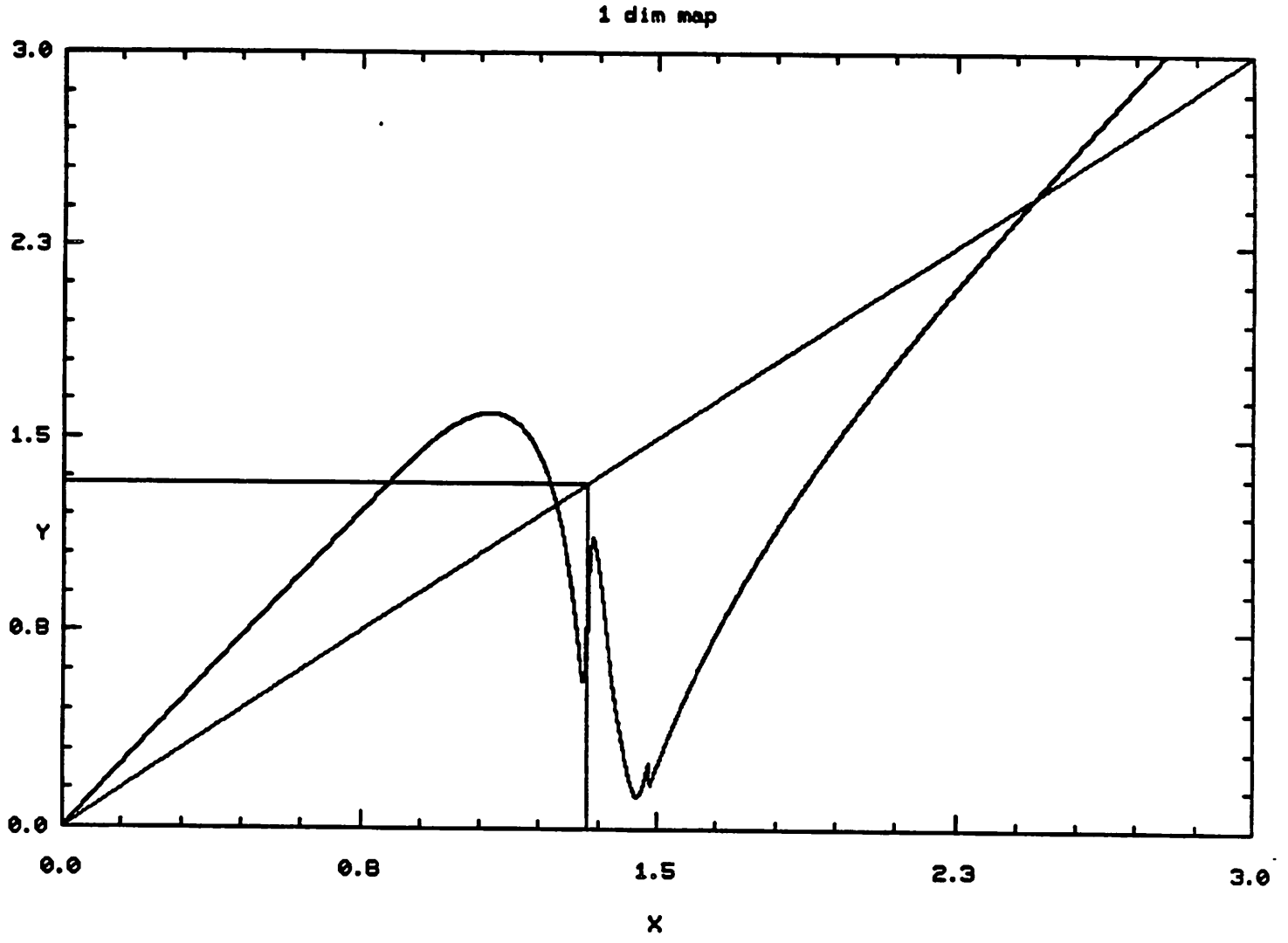

Fig. 5

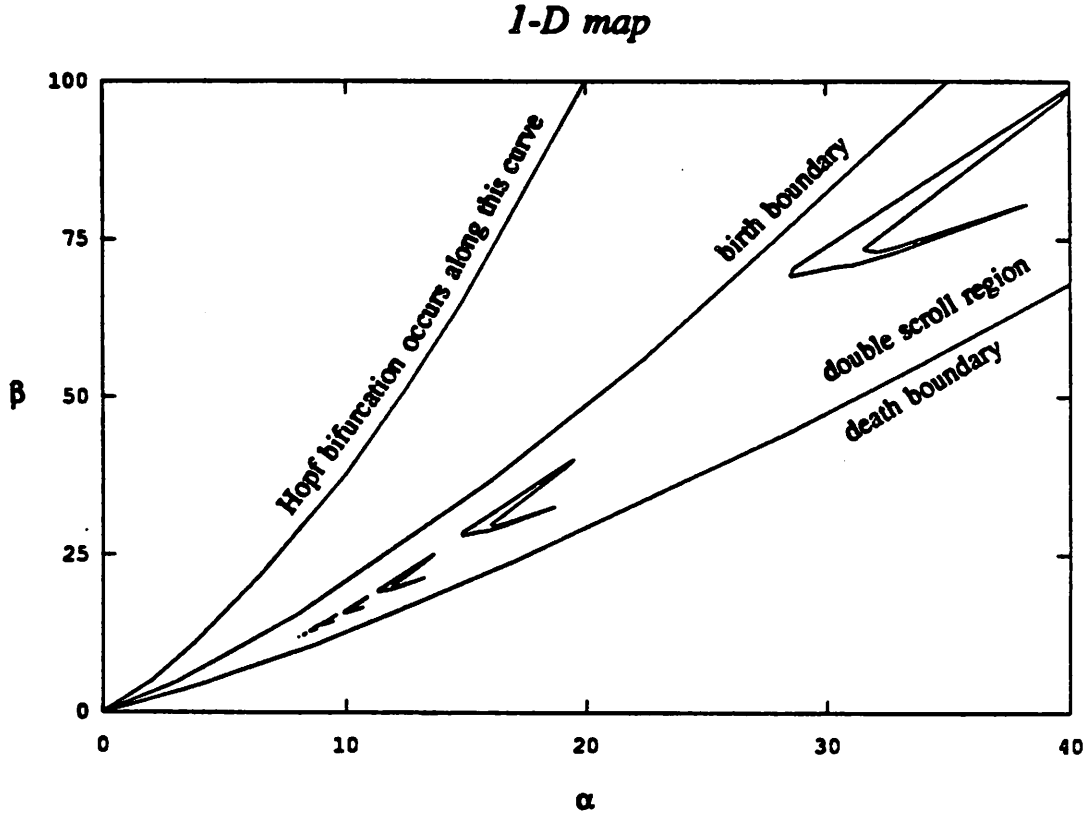
1-D map
100
75
50
25
0
β
0
10
20
30
40
α
Hopf bifurcation occurs along this curve
birth boundary
double scroll region
death boundary

Fig. 6

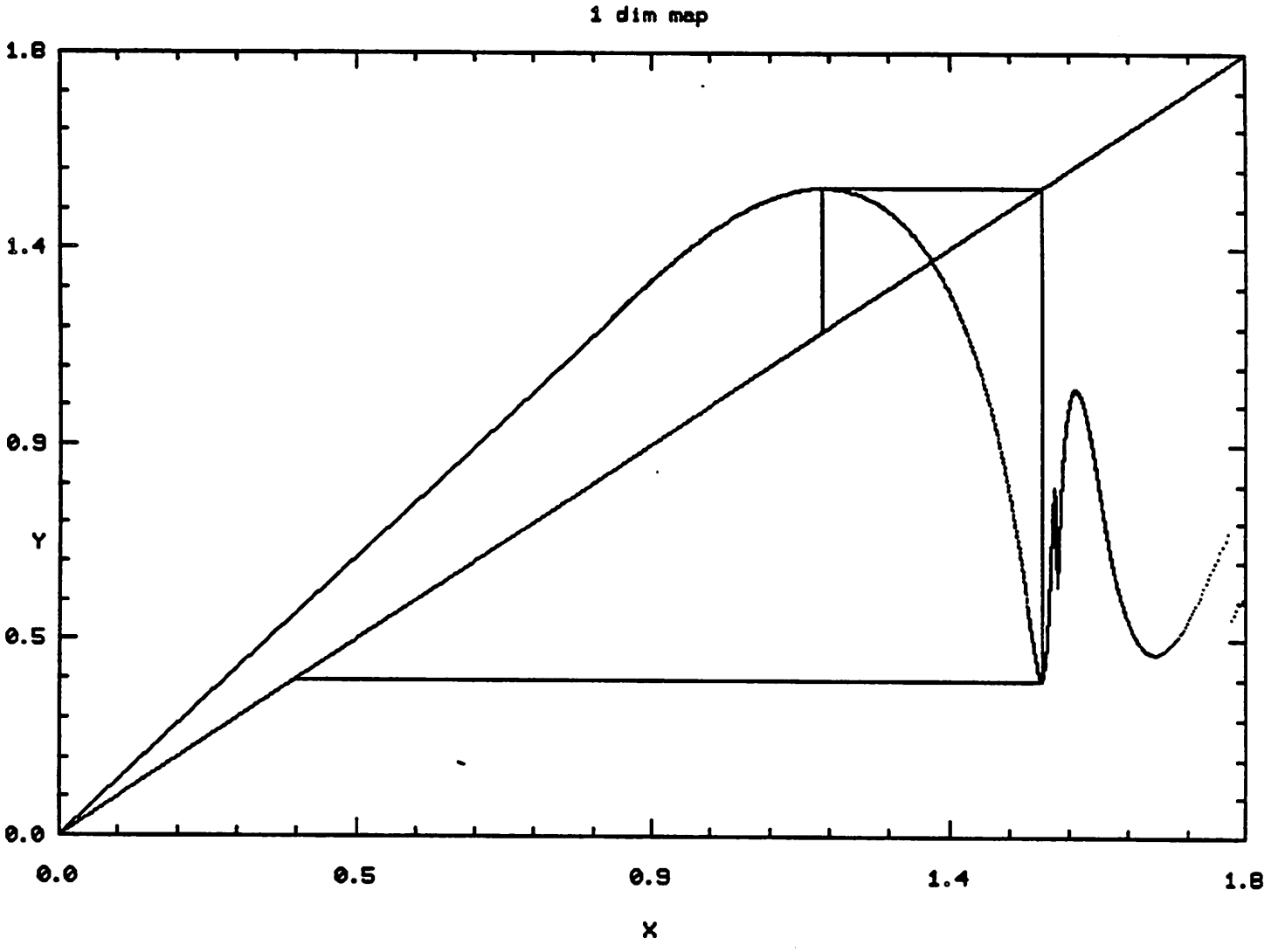

Fig. 7

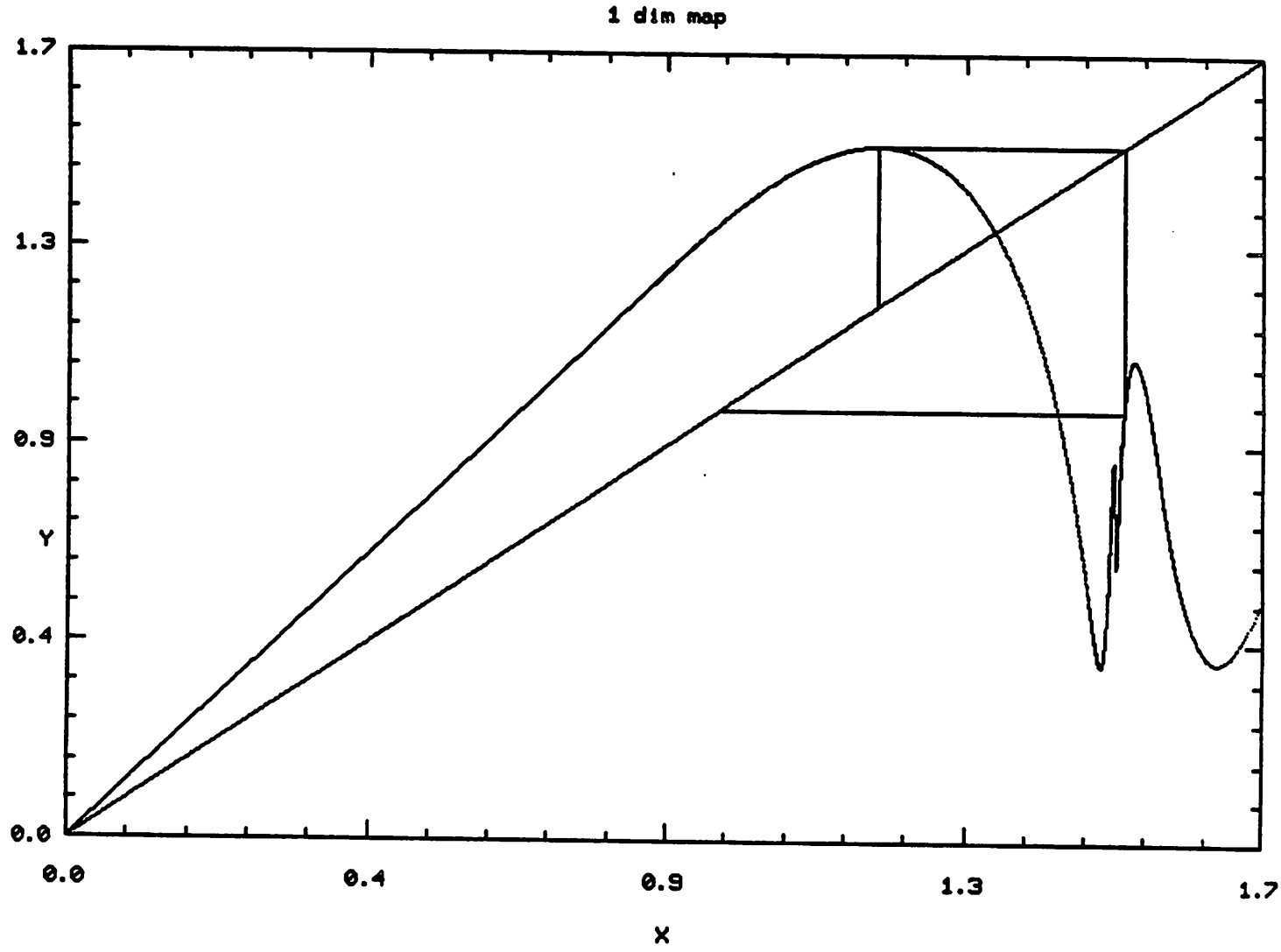

Fig. 8

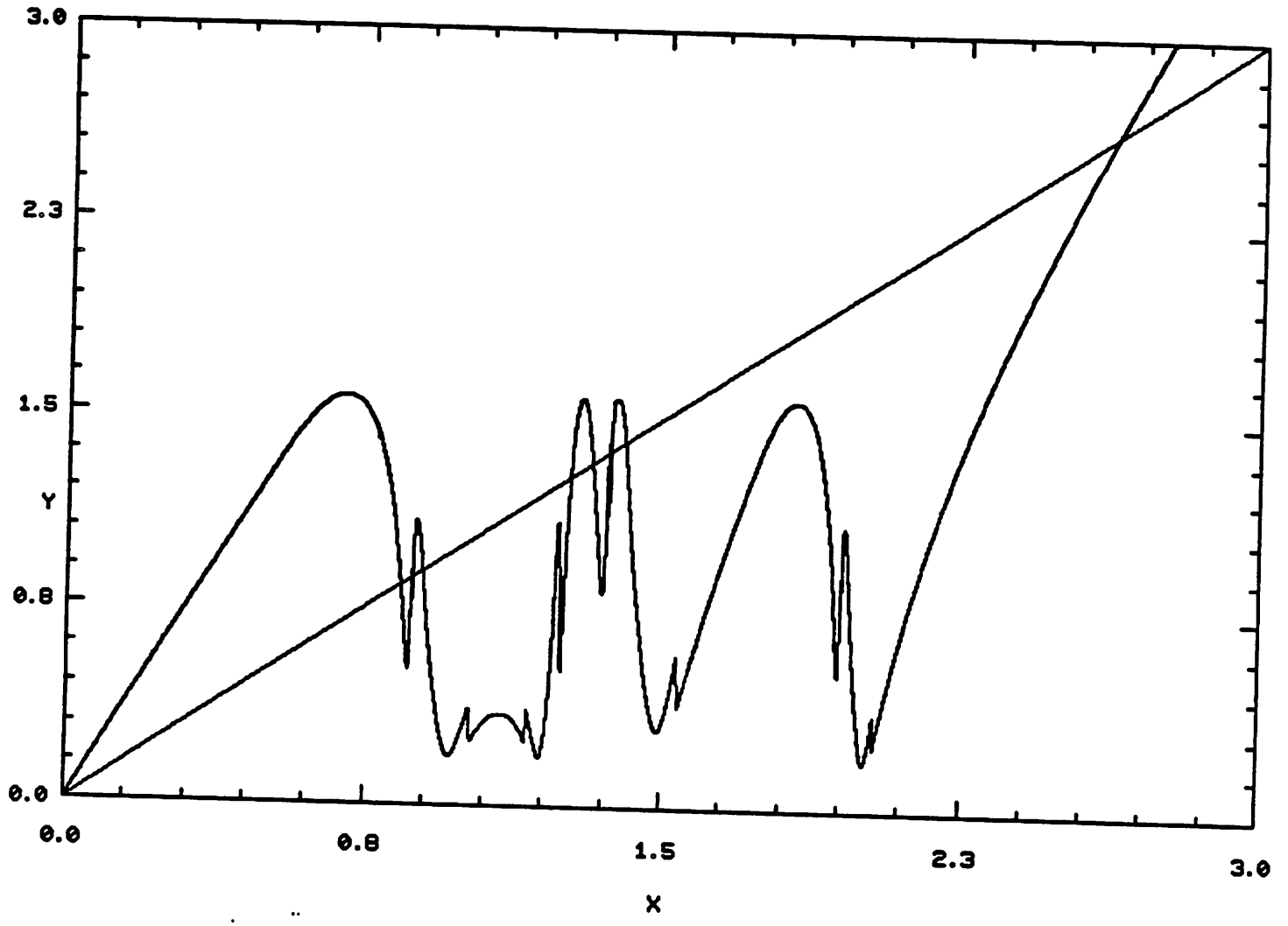

Fig. 9

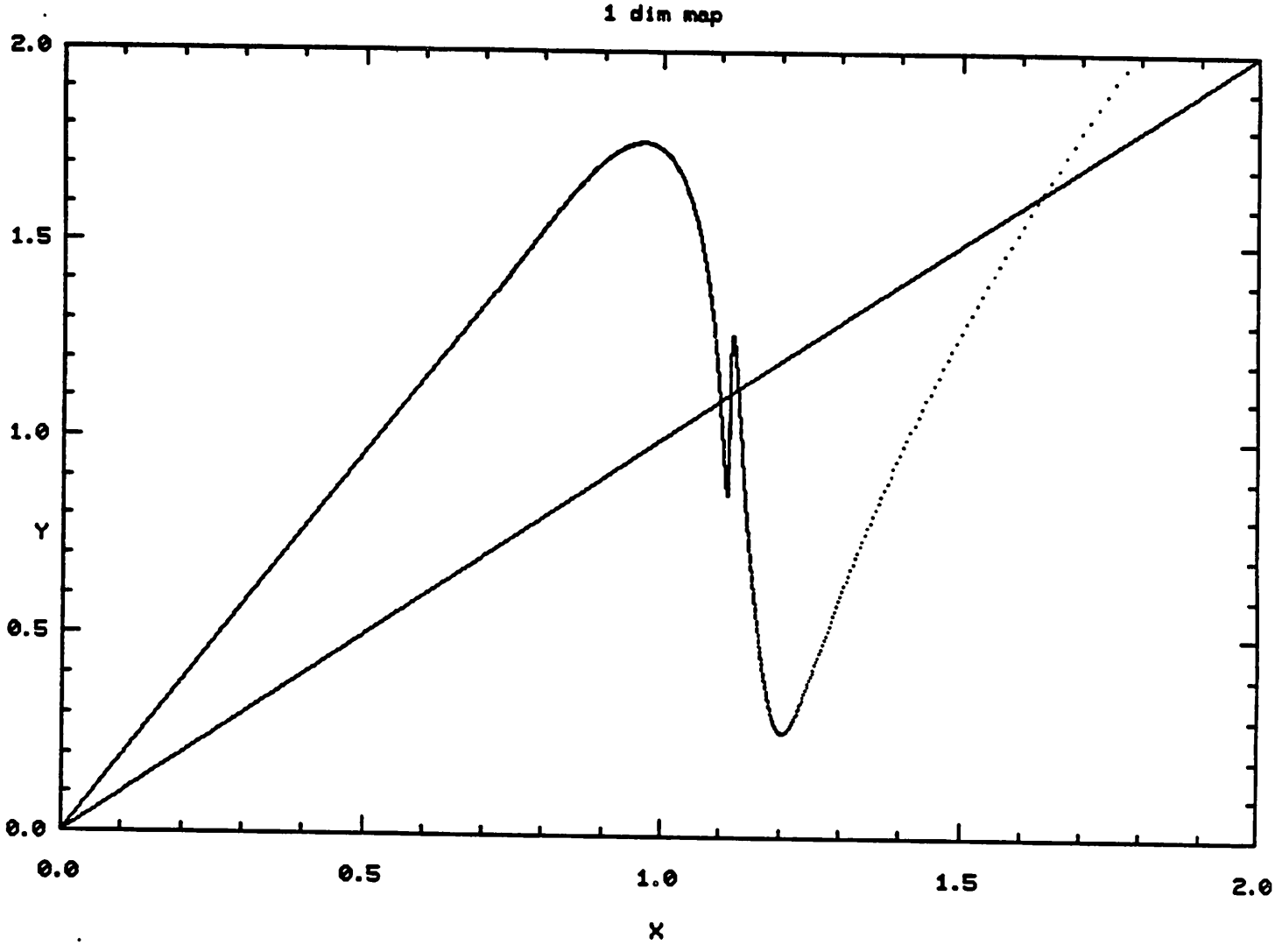

Fig. 10

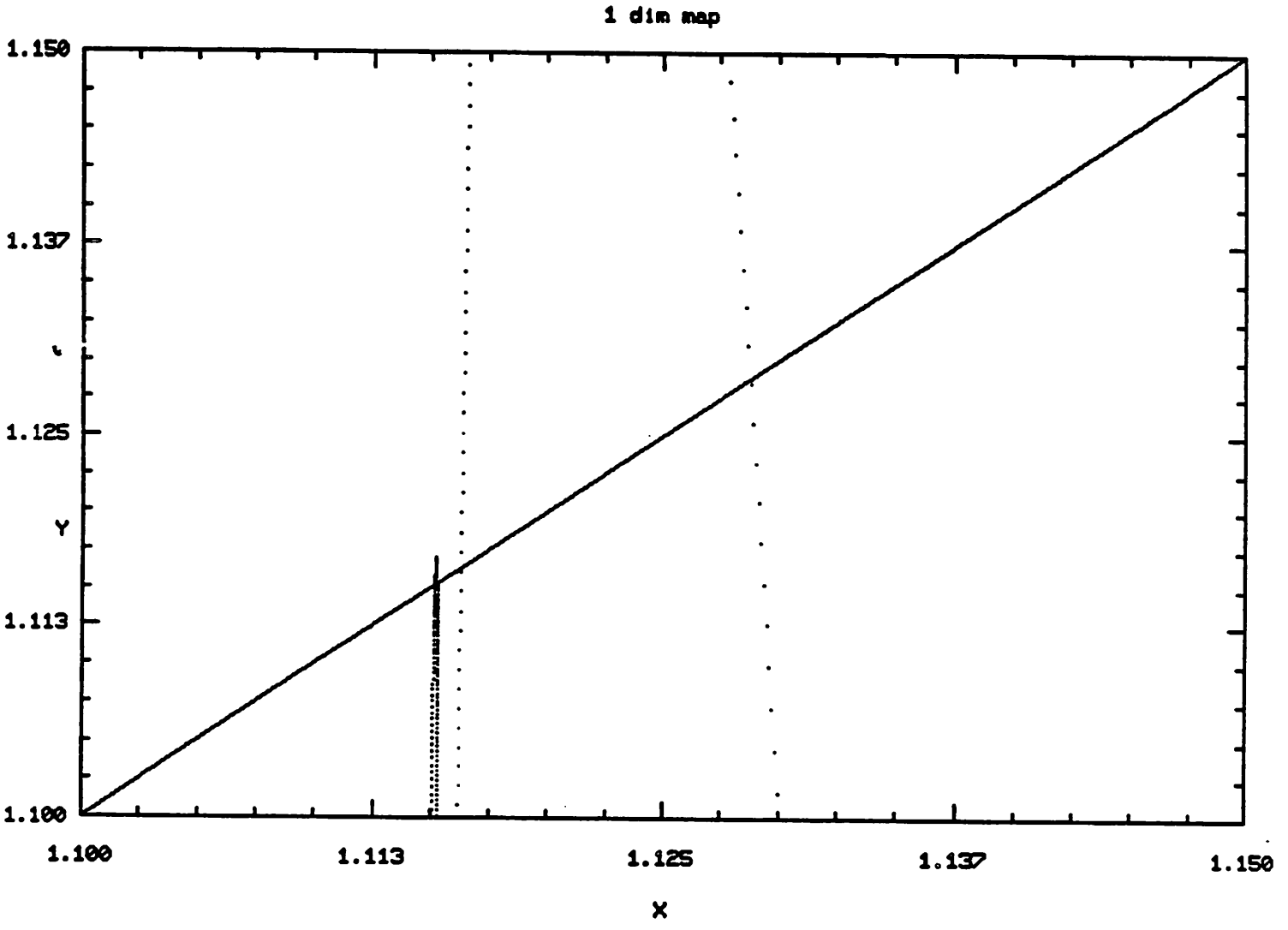

Fig. 11

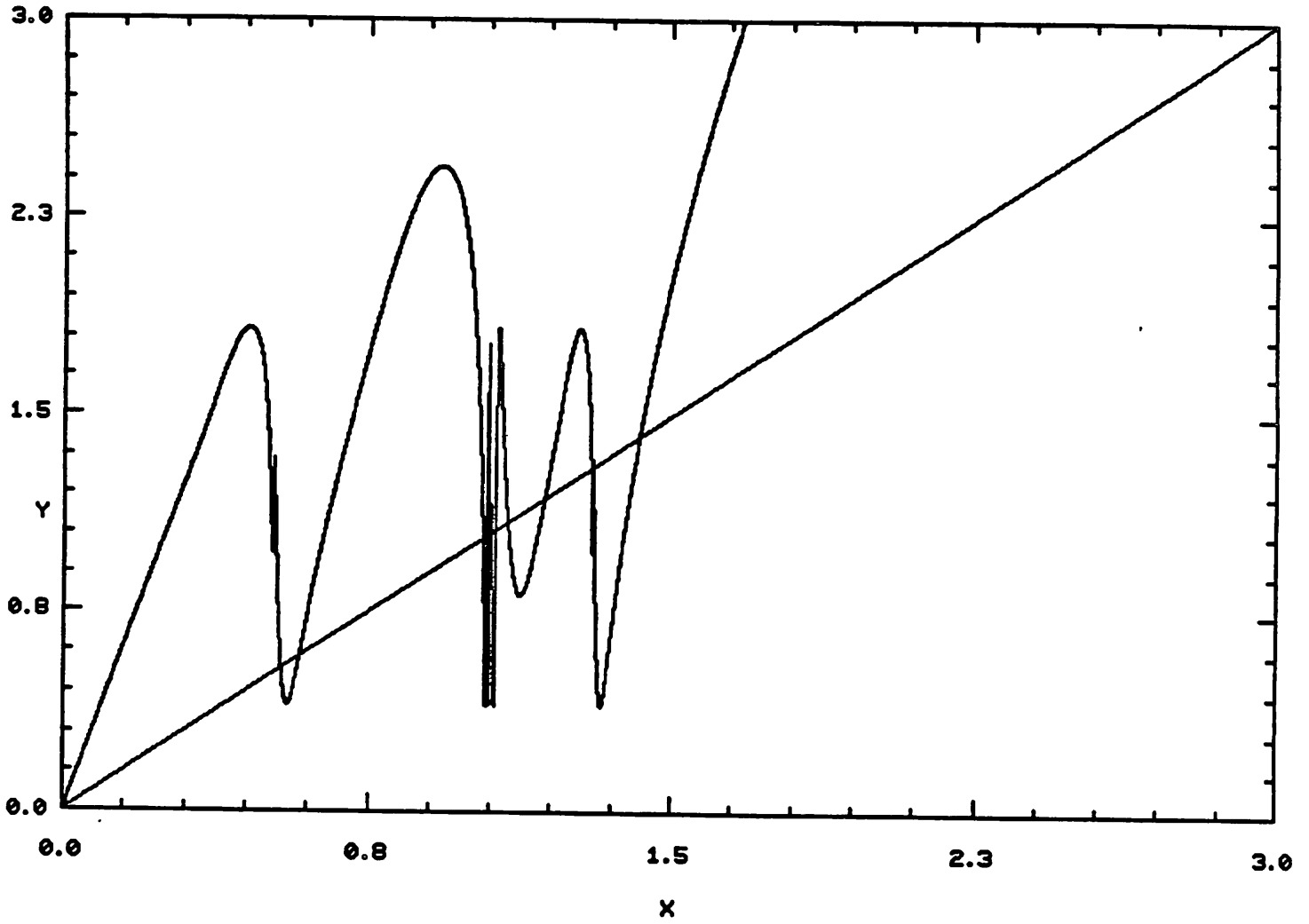

Fig. 12

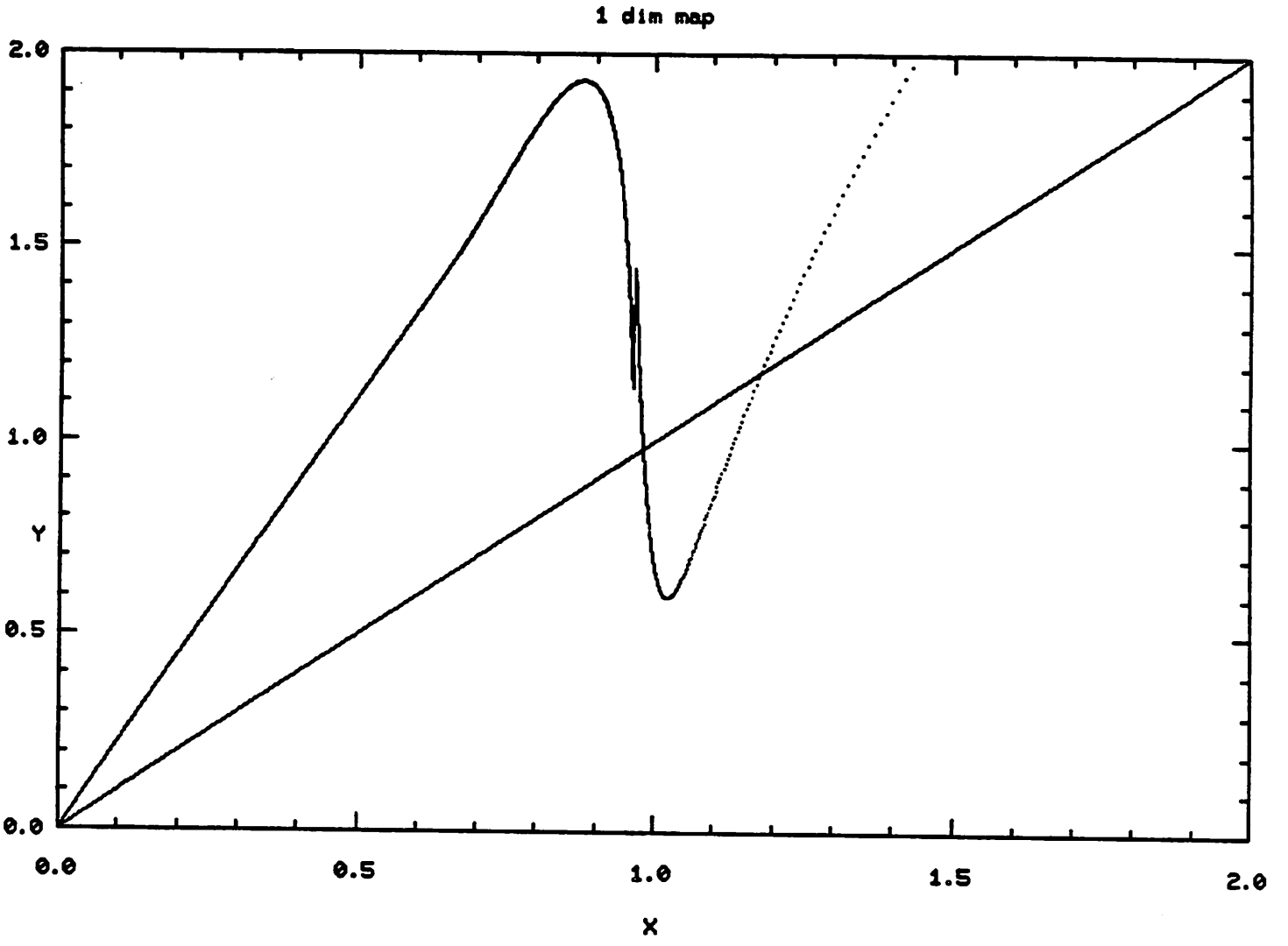

Fig. 13

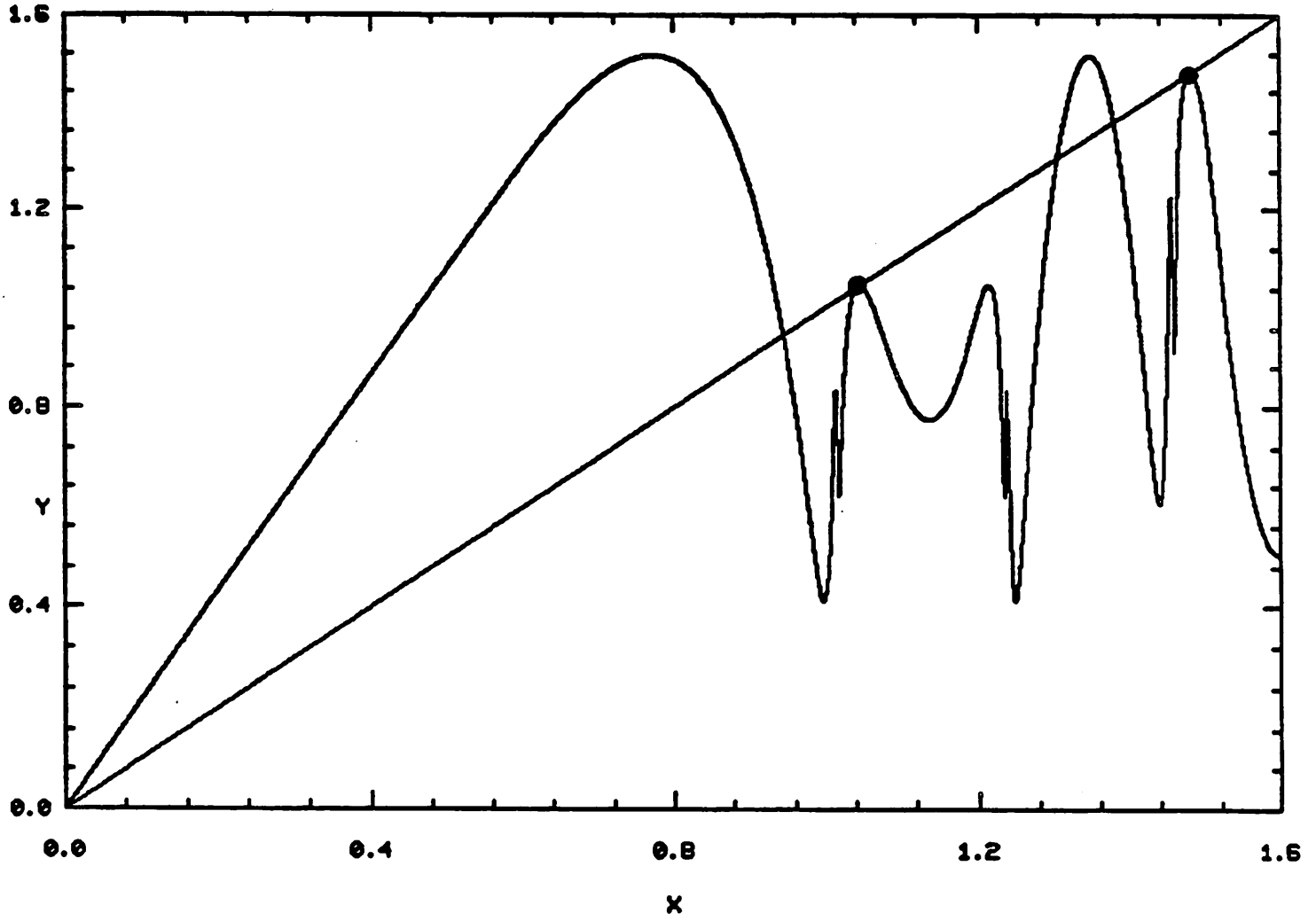

Fig. 14

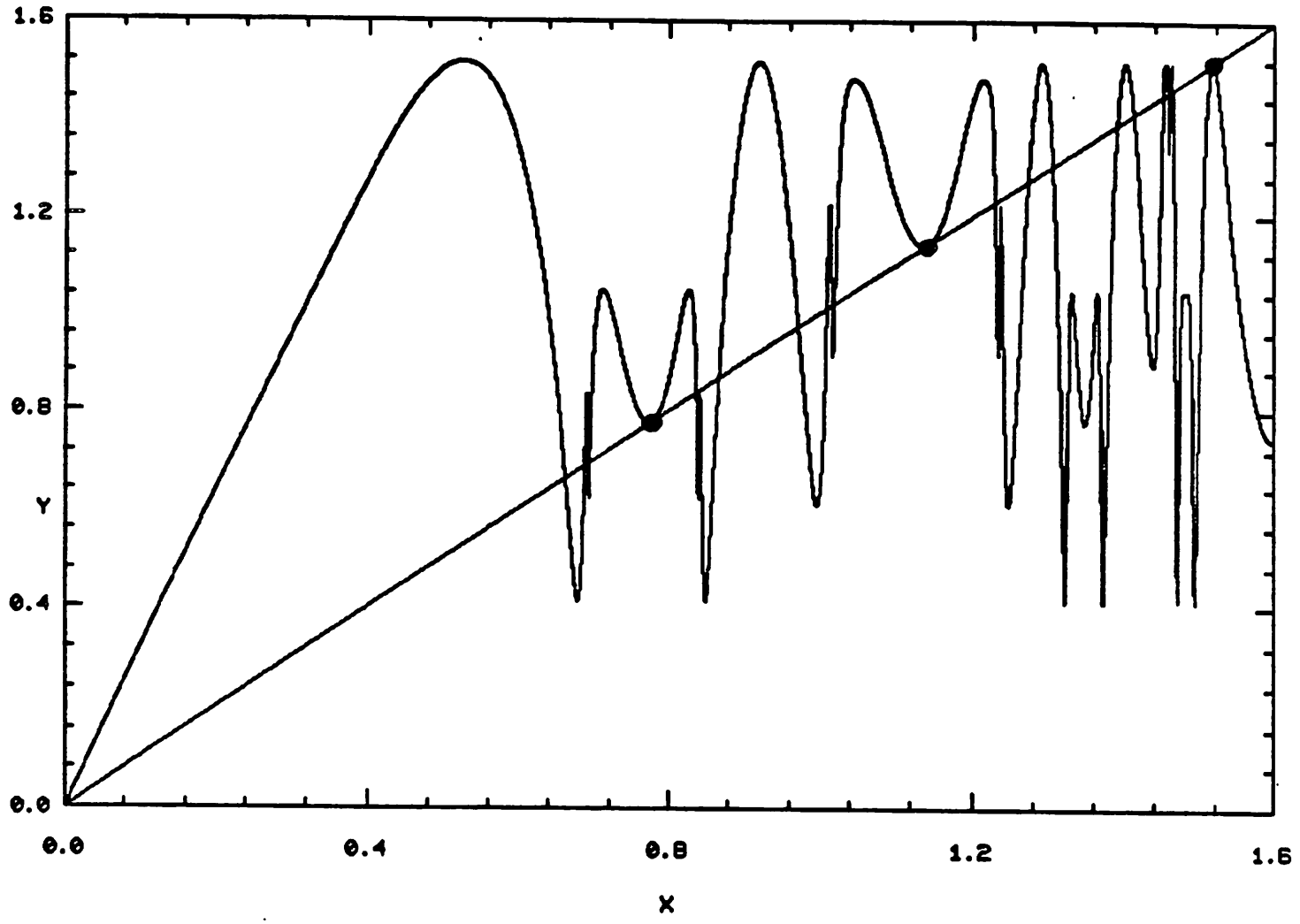

Fig. 15

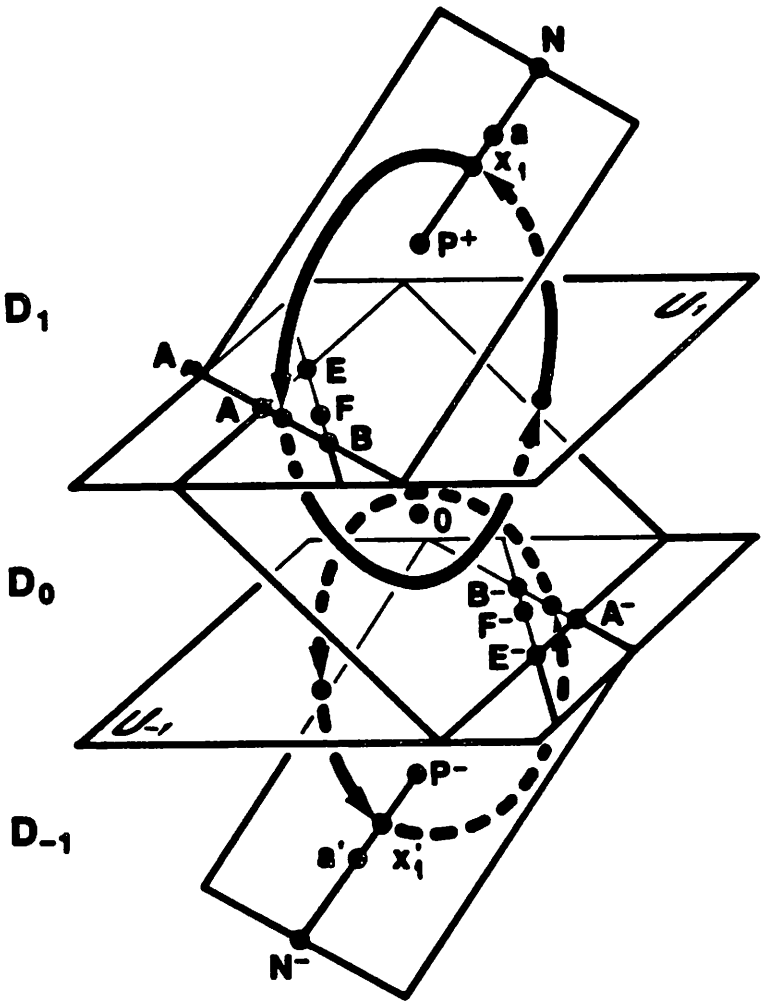

Fig. 16 (a)

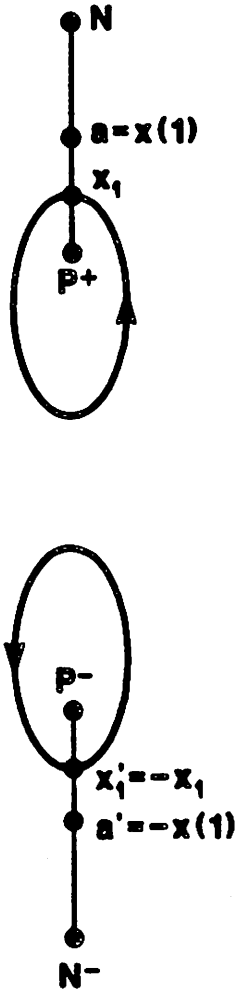

Fig. 16 (b)

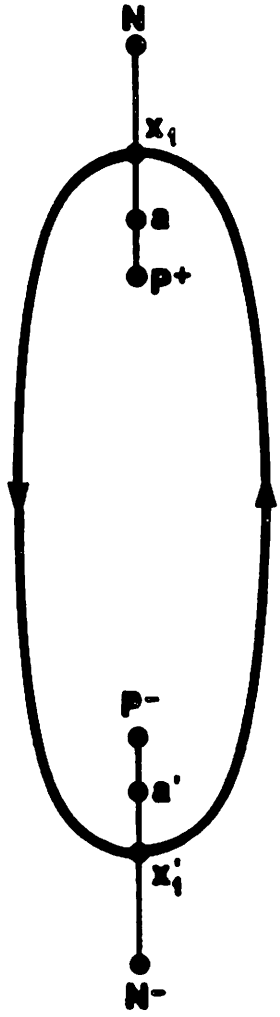

Fig. 17

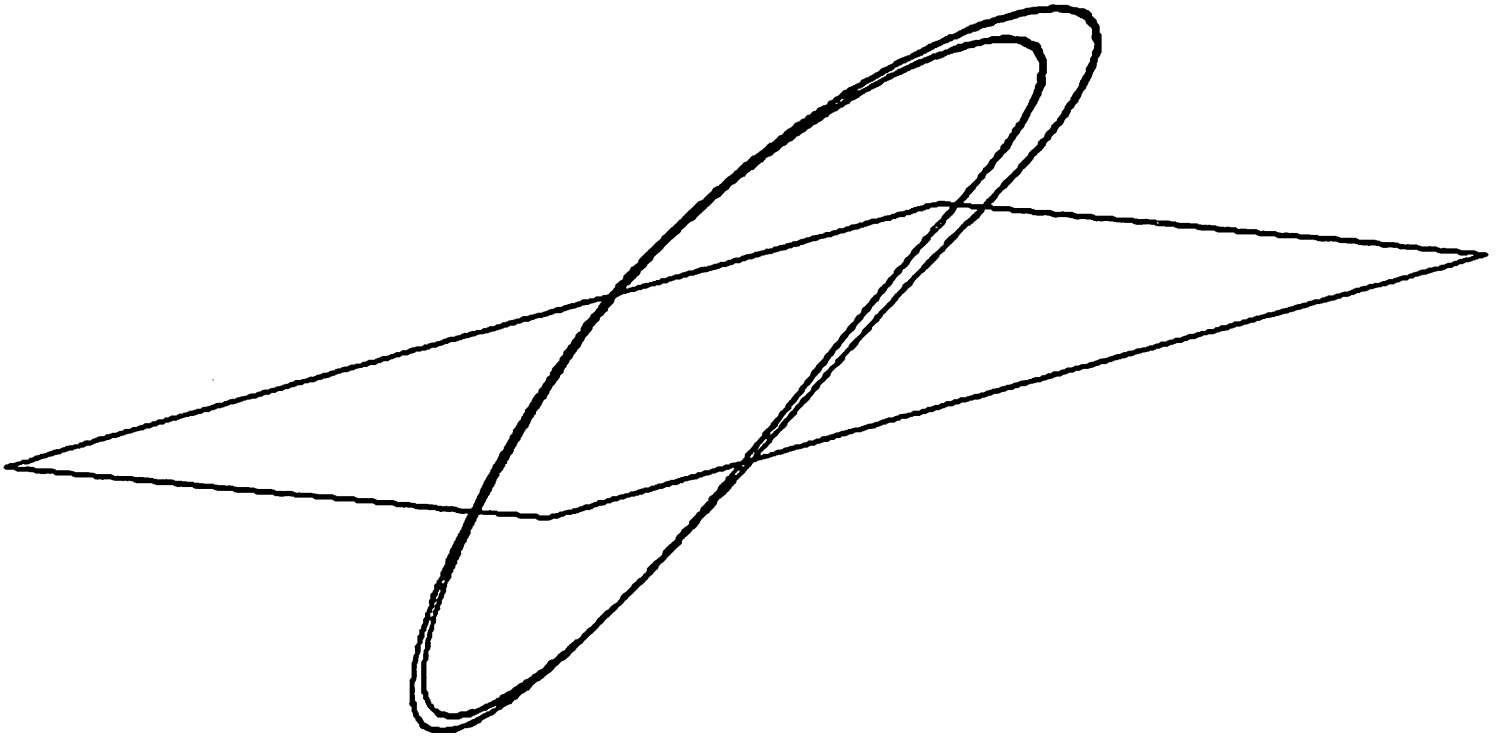

Fig. 18 (a)

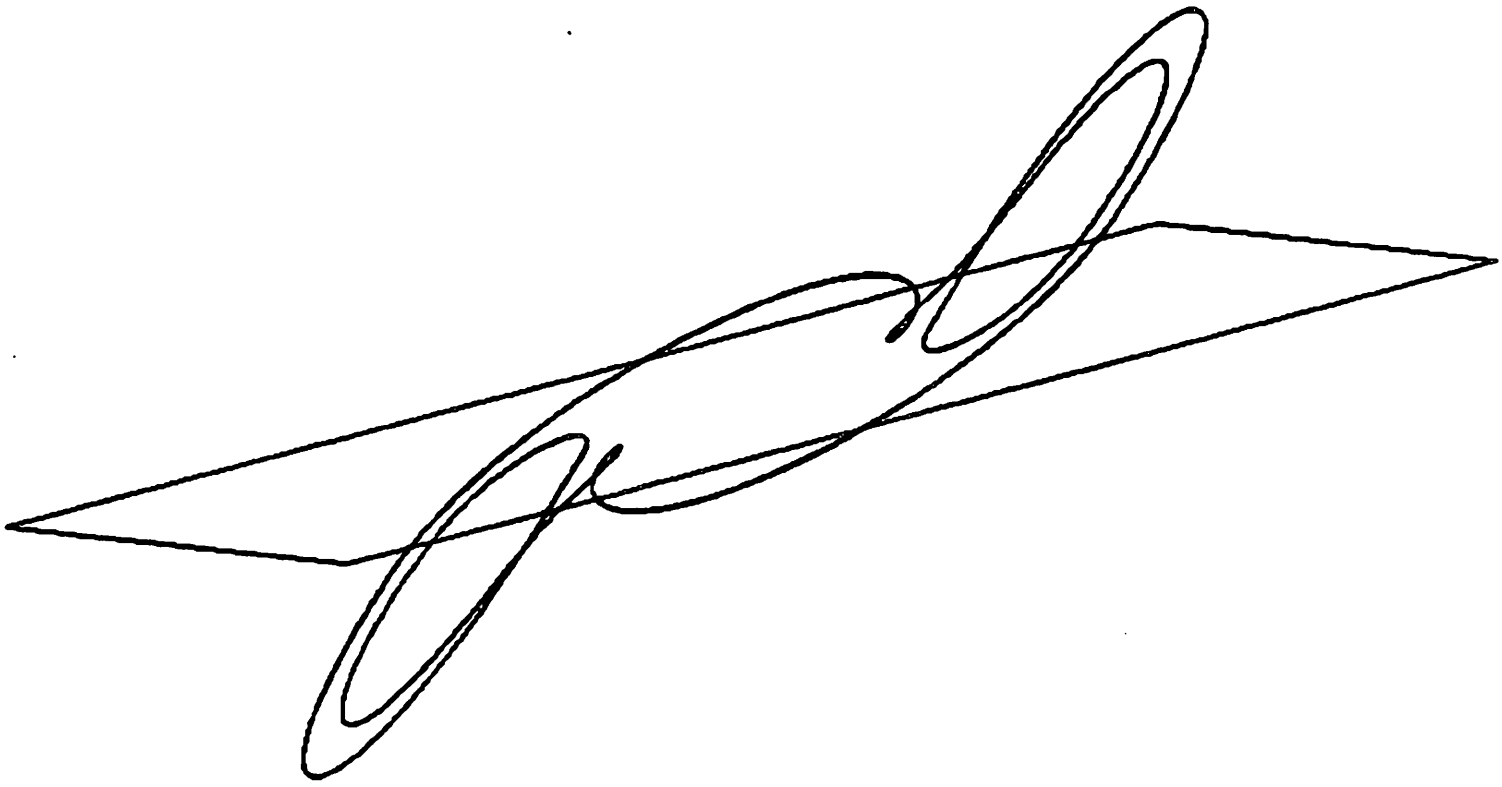

Fig. 18 (b)

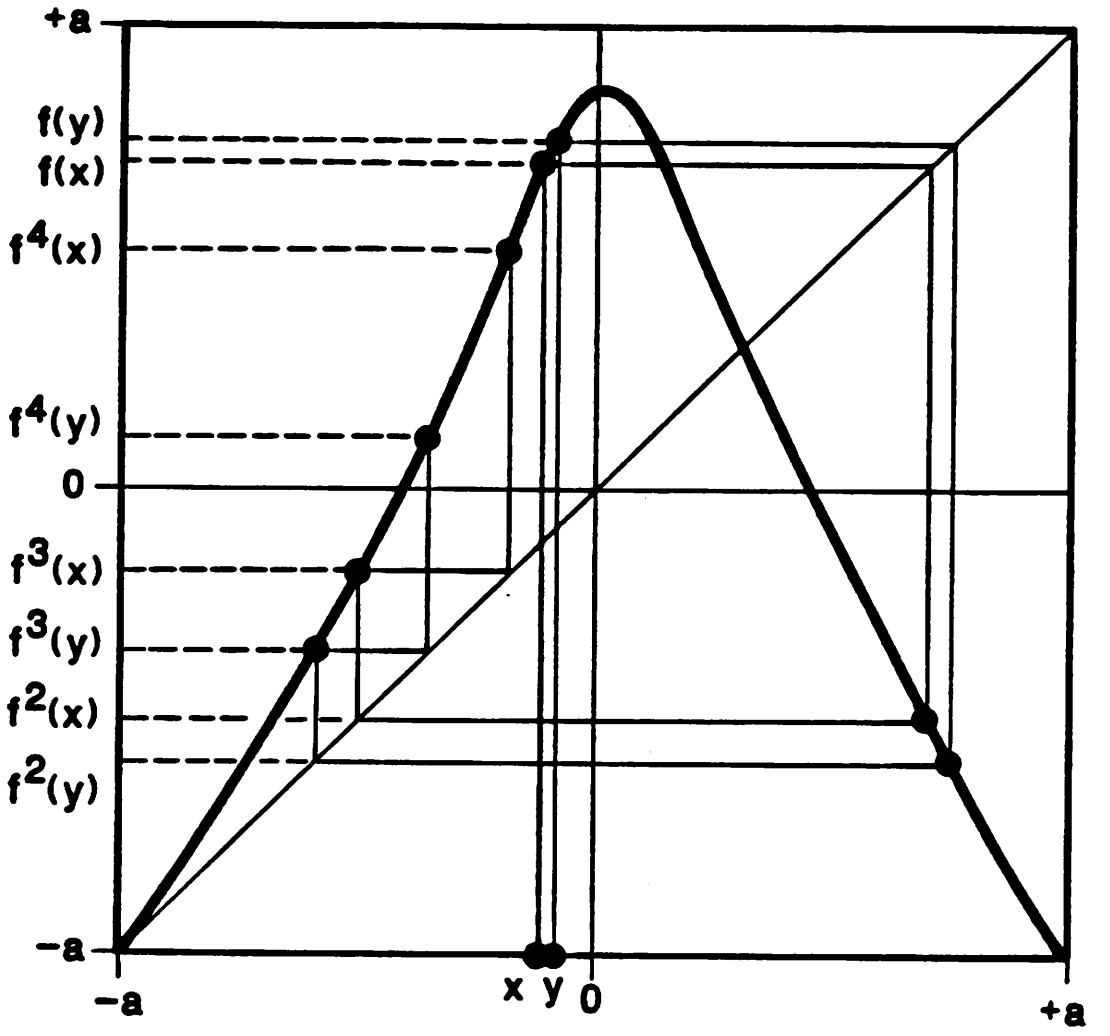

Fig. 19