İSTANBUL TEKNİK ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

NOKTASAL GİRDAP DİNAMİĞİ ÜZERİNE

YÜKSEK LİSANS TEZİ

Müh. Ahmet Kutsi NİRCAN

Ana Bilim Dalı : GEMİ İNŞAAT

Programı : GEMİ İNŞAAT

TEMMUZ 1992

ISTANBUL TECHNICAL UNIVERSITY * INSTITUTE OF SCIENCE AND TECHNOLOGY

ON THE POINT VORTEX DYNAMICS

M.Sc. THESIS

Ahmet Kutsi NİRCAN, B. Sc.

Date of Submission : June 22, 1992

Date of Approval : July 6, 1992

Advisor : Asst. Prof. Dr. Ali Rana ATILGAN

Jury Members : Prof Dr. M. Cengiz DÖKMECİ

: Assoc. Prof. Dr. Ömer GÖREN

July 1992

ACKNOWLEDGEMENTS

In what follows, a period of life, involving questions on Lagrangean formulation of point vortices, is reflected. During this period my advisor Dr. Ali Rana Atılgan's help and patience is greatly acknowledged. He taught me the basics of scientific research and was always there when I needed help.

During the preparation of this opus, Prof.Dr. M. Hasan Boduroğlu has kindly allowed the author to use the computational facilities of his projects. This help is gratefully acknowledged. Prof. Dr. Vural Cinemre's help on some aspects is greatly appreciated. Dr. Gazanfer Ünal's lecture notes greatly contributed to my understanding and his readiness for help beyond class is thankfully acknowledged. I would like to recognize Dr. İsmail Hakkı Helvacıoğlu for supplying me some necessary books for my thesis.

Special thanks goes to my dear sister Ayşin Nircan for her patience and support during the completion of this thesis.

I would like to thank Mrs. Ayşe Nazan Akman Özlüer for reading and editing my terrible sentences.

CONTENTS

LIST OF ILLUSTRATIONS

SUMMARY

Point vortex motion is introduced and application areas are discussed. Formulation for bounded and unbounded point vortices are given via Lagrangean representation. Also, the equations for advected particle in a system of point vortices are given.

Point vortex problem has integrable and nonintegrable solutions. In order to investigate nonintegrable solutions, an introduction to dynamical system analysis is given. Some tools to characterize the solution of dynamical system are explained. As an example Poincaré sections of Henon-Heiles system are drawn.

Nonintegrable dynamical systems are mainly investigated using computers. A new approach, object oriented programming, introduced and its application to this problem is explained. Lyapunov exponents program are discussed in more detail.

Numerical experiments are made for the unbounded, circular, and semi-circular domains. Configurations from literature reviewed and repeated. Also, new configurations are investigated. It is found that in the unbounded domain four, in a circular domain three, and in a semi-circular domain two vortices are enough to observe chaotic behavior. For all configurations, trajectories and power spectra, and for chaotic motions Poincaré sections are displayed.

Onsager's negative temperature paradox is discussed and recently suggested remedy to this paradox is given. Ergodicity of the point vortex systems are discussed. In the unbounded domain, four point vortex systems is made and it is found to be ergodic. For the bounded case billiard type chaos is ergodic since it resembles elastic collision of hard spheres.

NOKTASAL GİRDAP DİNAMİĞİ ÜZERİNE
ÖZET

Doğanın düzen içinde bir karmaşa içerdiği çevremizdeki olaylardan görülmektedir. Bu karmaşa birçok olayın gelecekteki davranımı hakkında tahmin yapmayı güçleştirmektedir. Örneğin günümüzde halen bir hafta sonraki hava durumunu doğru olarak tahmin etmek mümkün olmamaktadır.

İnsan bu problemi çözebilmek için doğal olayların modellerini yapmak ve bu basitleştirilmiş modeller yardımıyla doğayı anlama yoluna gitmiştir. Genellikle bu modellerde, modeli zorlaştıracak seçkisiz terimler ihmal edilmektedir çünkü bu tip terimler doğadaki karmaşayı modele de taşıyıp onu anlamamızı zorlaştırabilmektedirler. Fakat tamamen deterministik şartlarla oluşturulan modellerde de karmaşık davranımlar olabileceği matematikçiler tarafından ortaya atılmıştı. Bu durum diğer bilim dallarında 1960'lı yıllara kadar unutulmuştu ve deterministik modellerin karmaşık davranım gösterebilecekleri düşünülmüyordu (Gleick, 1990).

Lorenz (1963) bu yıllarda yayınlanan bir makalesinde oldukça basitleştirdiği bir atmosferik olayın modelinin kaotik davrandığını gösterdiğinde unutulmuş olan bu özellik tekrar hatırlanmaya başlandı. Lorenz'in yaptığı üç serbestlik dereceli adi diferansiyel denklem takımı konveksiyon yoluyla hareketi modellemekteydi ve kendini tekrar etmeyen bir yapıya sahipti. Bu sistemin önemli özelliklerinden biri de başlangıç şartlarına hassas olmasıydı. Başlangıç şartlarındaki ufak bir değişim daha sonraki hareketin tamamen farklı olmasına neden olmaktaydı.

Tekrar gündeme gelen deterministik sistemlerde kaotik davranım bilgisayarın kullanılmasıyla incelenebilir hale gelmişti. Lorenz'in makalesinden sonra bir çok bilim dalında benzer makaleler birbirini izledi, bu davranımı incelemek için yeni matematiksel yöntemler ortaya atılmaya başlandı.

Akışkanlar mekaniğinde karmaşık bir yapıya sahip olan türbülanslı akım doğadaki kaotik yapılara bir örnek olarak gösterilebilir. Basit deterministik sistemlerinde kaotik davranım göstermesi türbülanslı akımın anlaşılmasında yararlı olabilecektir (Ruelle, 1989).

Bu çalışmada kaotik davranım gösterebilen noktasal girdap sistemi, dinamik sistem analizi yöntemleriyle incelenmiş ve istatiksel mekaniğin burada kullanılıp kullanılamacağı, ergodiklik testi ile incelenmiştir.

Noktasal girdap, düzlemdeki girdap hareketinin çapının sıfıra indirilirken şiddetinin aynı kalması ile elde edilir. Bu hareket parçacıkların etkileşimi hareketi olduğundan atomik düzeyden kozmik düzeye kadar pek çok olaya benzeşim göstermektedir. Akışkanlar mekaniğinde önemli bir yeri olan girdaplar, akışkan içinde hareket eden cisimlerin arkalarındaki izde de görüldüğünden gemi inşaat mühendisliğinde de uygulama alanına sahiptir.

Dinamik sistemlerde kaosun incelenmesinde bilgisayarın önemi çok büyüktür. İncelemeler genellikle sayısal analiz yöntemleriyle yapılmaktadır. Zaman serileri incelemede ilk sırayı alırlar. Kaotik sistemlerde zaman serileri düzensiz bir görünüm sergilemektedirler. Zaman serilerinin Fourier serileri yardımıyla frekans tabanında incelenmesi sistemin periodları hakkında bilgi vermektedir. Spektrumda sürekli ve düzensiz bir grafik kaotik hareketin göstergelerindendir.

Bir diğer kıstas ise Poincaré kesitleridir. Poincaré kesitleri incelenen sistemin boyutunu indirerek incelemede kolaylık sağlarlar. Faz uzayında tanımlanan bir hiperdüzlemle kesişen yörüngelerin oluşturduğu noktalar Poincaré kesitlerini verirler. Bu yöntemle sistemin boyutlaını ikiye kadar indirmek mümkün olmaktadır. Bu kesitlerde dağılmış görünümlü noktaların varlığı kaosu haber veren bir başka bir belirtidir.

Kaosun en önemli kanıtlarından biri Lyapunov üstleridir. Lyapunov üstlerinin fiziksel anlamı sonsuz küçüklükteki bir kürenin yörünge boyunca geçirdiği deformasyonun ölçütü olarak açıklanabilir. Eğer sistemin en az bir tane pozitif Lyapunov üstü varsa bu sistem kaotiktir denilebilir. Pozitif Lyapunov üstleri sistemin sacıldığını, negatif Lyapunov üstleriyse sıkıştığını gösterirler. Sistemin saçılımı başlangıç şartlarının bir süre sonra unuttuğu anlamına gelmektedir ki, bu da kaos'un en önemli özelliklerinden biridir.

Yukarıda bahsedilen programlar yapılırken yeni bir programlama tekniği olan nesne tabanlı programlama kullanılmıştır. Nesne tabanlı programlama mevcut programlama dillerinden bazılarında da kullanılabilen fakat nesne tabanlı dillerle daha etkili olan bir tekniktir. Bugüne kadar kullanılmakta olan işlem tabanlı programlama tekniğinde yapılan işlemler daha önemli bir yer tutmaktaydı. Oysa nesne tabanlı programlamada nesneler daha önemli yer tutmaktadır. Bu teknikte her nesneye

kendinden yapılması istenen metodlar öncelikle programlanır ve gerektiğinde sadece nesneden bu yöntemi yapması komutu verilir. Böylece programda her seferinde yapılacak şeylerin yinelenmesi önlenmiş olmakta ve programın dizaynı ve anlaşılması daha kolay olmaktadır. Nesne tabanlı programlamanın bir diğer önemli yararı ise yaratılan nesnelerin her yeni yazılan programa kolaylıkla uygulanabilmesidir. Bu özellik sayesinde program yazma zamanı kısalabilmektedir.

Bu çalışmada bir nesne tabanlı dil olan C++ dili kullanılmıştır. C++ dili çok yaygın olarak kullanılan C diline nesne tabanlı programlamayı kolaylaştıracak bazı ekler yapılmasıyla oluşmuştur ve C dili ile tamamen uyumludur.

Çalışmada yararlanılan programlarda kullanılmak üzere bir takım nesneler yaratılmıştır. Yaratılan küme nesnesi kendini oluşturan sayıları ve boyutunu bilen bir nesnedir. Bu nesne bilgisayarın hafıza kontrolü gibi bazı temel işlemleri ve iki küme değişkenin birbiri ile karşılaştırma işlemlerini içeren metodları bilmektedir. Nesne tabanlı dillerin bir özelliği olan kalıtım kullanılarak küme nesnesinden vektör, matris, hiperdüzlem nesneleri türetilmiştir. Bu nesneler de kendilerine özgü metodlarla donatılmışlardır. Bu nesneler yanında adi diferansiyel denklemleri içeren bir nesne de yaratılmıştır.

Bu nesneler kullanılarak Runge-Kutta ve Bulirsch-Stoer sayısal integrasyon yöntemleri C++ diline uyarlanmıştır. Bu yöntemler sayesinde yörüngeler, spectrumlar ve Poincaré kesitleri çizilmiştir.

Lyapunov üstleri programları en çok kullanılan basit bir yöntem kullanılarak yapılmış fakat bu yöntemle üstlerin bulunmasının çok uzun zaman alacağı anlaşılmıştır. Daha kısa zaman alan yöntemler üzerindeki çalışmalar sürmektedir.

Noktasal girdaplarla ilgili incelemeler, sınırlandırılmamış ve sınırlandırılmış ortamlar için yapılmıştır. Her iki halde de sabit noktaların varlığı araştırılmıştır. Sınırlandırılmamış halde tek girdap, hareket denklemlerinin sağ tarafının sıfırlanması nedeniyle her başlangıç şartı için sabit nokta vermektedir. İki girdabın sabit nokta oluşturabileceği tek durum çözüm kümesinde bulunmayan iki girdabın üst üste gelme durumudur. Bu yüzden iki girdaplı sistemde sabit nokta bulunmamaktadır.

Girdap sayısının ikiden büyük olduğu her durumda sabit noktalar bulunmaktadır. Bu sabit noktalardan bazılarının karakterleri incelenmiştir. Üç noktasal girdapın oluşturduğu sabit noktaların stabil olmadığı görülmüştür. Dört noktasal girdabın oluşturduğu sabit noktaların ise stabil olduğu, sabit nokta veren başlangıç şartlarından

büyük sapmalarda yörüngelerin sabit noktalar etrafında kalmadığı gözlenmiştir. Sabit noktaları incelenen son hal ise beş noktasal girdaptır. Bu halde sabit nokta veren konfigürasyondan yapılan çok küçük sapmaların dahi yörüngelerin bu noktalardan uzaklaştırdığını göstermiş ve beş noktasal girdabın sabit noktasının stabil olmadığı bulunmuştur.

Sınırlandırılmamış ortamda kaotik hareket elde etmek için en az sayı olan dört girdap kullanılmıştır. Dört girdap simetrik başlangıç şartları verildiğinde en kötü halde kuasi-periyodik hareket etmektedir. Sınırlandırılmamış ortamda bir diktörtgenin köşelerine yerleştirilen noktasal girdapların hareketi bir tor oluşturmaktadırlar. Bu başlangıç şartından yapılan küçük sapmalar sonucunda KAM teoremi uyarınca yörüngeler yine bu torun üzerinde kalmaya devam etmektedir. Daha büyük sapmaların kaotik davranım gösterdiği görülmüştür.

Bir eşkenar üçgenin köşelerine ve açıortayların kesişim noktasına dört noktasal girdapla oluşturulan simetri halinde, eğer girdapların şiddetleri benzer ise, köşelerdeki noktasal girdapların dairesel bir yörüngeyle döndüğü ve ortadaki noktasal girdabın ise sabit kaldığı görülmüştür. Bu başlangıç şartında yapılacak ufak bir sapma hareketi kaotik yapmaya yetmektedir.

Dört noktasal girdapla yapılabilecek bir diğer simetri ise noktasal girdapların bir çizgi üzerinde yerleştirilmesi ile elde edilebilir. Bu durumda ilginç bir olayla karşılaşılmıştır. Çizgi üzerindeki noktasal girdaplardan içteki çiftin birbirine olan uzaklığı dışardakilerle olan uzaklığın yarısı alınmış ve şiddetleri $+1$, $+1$, -5 ve -5 olan noktasal girdaplar mümkün olan dört şiddet konfigürasyonunda incelenmiştir. Her dört konfigürasyonda da düzgün hareket beklenmesine karşın şiddetleri küçük olan noktasal girdaplar dışarıya konulduğunda sistemin kaotik davrandığı gözlenmiştir. Bu tip bir davranıma literatürde rastlanamamış ve mevcut yöntemlerle sebebi henüz açıklanamamıştır.

Sınırlandırılmamış ortamda aynı değerde fakat ters şiddetteki noktasal girdaplar birlikte sonsuza doğru hareket etmektedirler. Bu hareket örnek olarak iki pozıtıf iki negatif noktasal girdabın hareketi verilmiştir.

Sınırlandırılmış halde noktasal girdaplar dairesel ve yarı dairesel ortamlarda incelenmiş ve bu tip bir ortam için sabit noktalar verilmiştir. Dairesel ortamda kaotik davranım elde etmek için gerekli en az noktasal girdap sayısı üçtür. Üç noktasal girdapla elde edilen kaotik davranıma iki örnek gösterilmiştir. Bunun yanında bilardo tipi kaos adı verilen bir davranım dairesel ortamda dört noktasal girdapla gösterilmiştir.

Sınırlandırılmış haldeki diğer incelemeler yarı dairesel ortamda yapılmıştır. Yarı dairesel ortamda kaos elde etmek için gerekli olan iki noktasal girdaplı konfigürasyonlar incelenmiştir. Stabil sabit nokta veren bir konfigürasyona yapılan sapmalar sonucunda oluşan kuasi periyodik ve kaotik davranımlar gösterilmiştir. Ayrıca yarı dairesel ortamda da bilardo tipi kaos adı verilen olaya bir örnek verilmiştir.

Normalde çok serbestlik dereceli sistemlerde kullanılan istatiksel mekaniğin, düşük serbestlik dereceli kaotik sistemlerde de kullanılabileceği bilinmektedir. İstatiksel mekaniğin kullanılabilmesi için gerekli olan ergodiklik şartı noktasal girdap için araştırılmıştır. Sınırlandırılmamış halde yapılan ergodiklik testi sonucunda kaotik noktasal girdap sisteminin ergodik olduğu görülmüştür.

Noktasal girdapların incelenmesi sonucunda Onsager (1949) tarafından ortaya atılan paradox ve bunun yakın zamandaki çözümü de (Berdichevsky, Kunin ve Hussain, 1991) yine son bölümde tanıtılmıştır.

CHAPTER 1
INTRODUCTION

Since the beginning of early ages man tried to comprehend the nature and dreamed to control its behavior. At first, it was thought that this could be done with the aid of simple models. It was thought, for example, that all the planets has perfectly circular orbits. Kepler himself believed this at first, but he found out that the orbits of planets were not circular. Despite this, and other similar observations, looking for perfectness and to trying to explain the nature with simple models continued.

However, it was soon understood that the nature has many complexities in its structure and these complexities make its behavior unpredictable in many situations. This is why modeling the nature is a difficult task. It is possible to take many factors into account in a mathematical model; however, doing this makes the model difficult to comprehend. Usually simplifications, and especially linearizations are used to avoid this kind of complexities. Linear models of systems helped science to solve many problems, however, as science and technology develop, scientists need to include more factors in their models.

The purpose of modeling is to understand and to predict the behavior of a system. Until 60's, behavior of models was expected to be predictable at all times and thus some scientists were thinking that they were very close to controlling the nature. Von Neuman, for example was talking about his plans to predict and even to control the weather at a global scale (Gleick, 1990). It was thought that predicting weather would be possible with large scale atmospheric observations and with the use of newly developing computers. However, Lorenz (1963) showed with computer experiments that even a very 'simple' model of atmospheric behavior was not simple at all. Lorenz's model had three degrees of freedom but its behavior was unpredictable. This showed

that to control and even only to predict the weather is not so easy. Lorenz's paper became very famous not only in his discipline but also in many areas of science and a new concept 'chaos' emerged. Following this development new tools to investigate chaos appeared and scientists began to use these tools to explain difficult or 'strange' behaviors that they encounter in their fields. Most of these tools require numerical approach and computers became a must to investigate chaos.

In fluid mechanics, first example of irregular motion that comes to mind is turbulence. Turbulent flow generates a noisy nonperiodic spectrum. It is thought that such a spectrum is due to irregular inputs to the system. Another explanation involves the presence of many oscillators. It had also been thought that these oscillators would have given quasi-periodic behavior. Recently, it has been shown that many oscillator systems could yield chaotic behavior even when they are deterministic (Ruelle, 1989). In the study of turbulence importance of chaos appears at this point.

In this work point vortex motion which is capable of displaying chaotic behavior will be investigated. With its capability of showing chaotic behaviors point vortex system can be a useful model to investigate two dimensional turbulence (Novikov and Sedov, 1978).

Background for vortex motion will be given and rectilinear point vortex will be explained in the following sub-section. After that, previous works and present approach will be given.

1.1. Background

Vortex motion is a three dimensional motion. In three dimensions, a vortex line is defined as follows: a line drawn from point to point such that its direction is everywhere that of the instantaneous axis of rotation is called a 'vortex line' (Lamb, 1932). If it is possible to draw vortex lines through every point of a small closed curve this is called a vortex tube. The fluid contained within such a tube is called vortex or

vortex filament.

When the motion is in two dimensions it is called rectilinear vortex (will be called point vortex hereafter). This time vorticity becomes perpendicular to the plane that we observe the motion and the vortex tube will intersect this plane to form a curve. If this curve is a circle, then it is possible to define the strength as follows

$$2\pi\kappa = \pi a^2 \omega \tag{1.1}$$

where a is the radius of circle. Point vortex motion results when κ is constant and a is shrunk to zero.

Since Stokes and Helmholtz it is known that any velocity field V can be split into a sum of two parts. One with the same divergence as V, and no curl, and one with the same curl as V and vanishing divergence (Gurtin, 1972). For incompressible flow, the first part is irrotational and divergence-free and thus leads to the linear problem of potential flow. The second part, however, derives directly from the velocity of the field V. Dynamics of this part is the basis of the point vortex problem, (Aref, 1983) since it is nothing other than a circular motion.

Point vortex problem ensembles many dynamical systems, from celestial mechanics to atomic level, since it is a equation of interaction of particles. In mathematics, it is convenient to expand the class of solutions by applying the smoothness requirement when searching for a general theorem. In this sense, point vortex is a useful model for two dimensional turbulence. Such vortices, play a fundamental role in superfluidity and superconductivity phenomena, and are widely used in the simulation of various non-smooth flows as well as of plasma in a magnetic field (Novikov and Sedov, 1978). Roughly speaking, integrable behavior corresponds to laminar flow, while non-integrable motion corresponds to turbulence. Therefore, studying non-integrable flow in the case of point vortex is the primary interest.

Like most dynamical systems, the Navier-Stokes equation is capable of displaying both integrable and non-integrable motions. Since notions of integrability and non-integrability are considerably clearer for two-dimensional systems of ordinary differential equations (ODE), then for the partial differential equations, such as the Navier-Stokes equation, it is, therefore, both appropriate and useful to distinguish flows as the ODEs (Aref *et al.*, 1989).

Chaotic behavior of dynamical systems is usually investigated using of computer experiments and it is known that Lagrangean representation is a better approach whenever computers are to be used. In Eulerian representation, velocity fields dependent on space and time are used. Whereas in Lagrangean representation, particles and their velocities used, just like Newtonian dynamics of a system of particles (Stuart and Tabor, 1990).

1.2. Previous Works

The identification of vortex motion as a subtopic within fluid mechanics goes back to the 1858 paper of Helmholtz (Aref and Kambe, 1988), and one of the most important early work on point vortex by Gröbli (Aref *et al.*, 1992). Since then the literature on this topic has grown steadily.

Novikov (1976) investigated point vortices as a model of two dimensional turbulence. In his paper, he gave some statistical properties of point vortices. Novikov and Sedov (1978) give numerical experiments and discuss mixing properties. Later, Novikov and Ledov (1980) investigated chaotic motions by displaying Poincaré maps.

Aref and Pomphrey (1982) investigated integrable and chaotic motions of four vortices of identical strengths in unbounded domain. They applied canonical transformation and reduced the number of degrees of freedom by two. With this reduced system they investigated three and four vortices by displaying trajectories and Poincaré maps for configurations largely perturbed from their symmetric rectangular

form.

Aref (1983) in his review paper discussed integrable and chaotic vortex motion in separate sections and the many vortex problem. The author stated that there is a threshold for chaotic behavior in point vortex systems. For a given flow geometry, there will be a maximum number of point vortices, N_*, so that regular motion can be observed. If the number of point vortices is $N > N_*$ the system will be chaotic. For unbounded motion, this threshold is 3, for a circular domain N_* is 2, and for a semi-circular domain $N_*=1$.

Aref (1984) discussed stirring in a circular domain with numerical experiments. Another paper discussing bounded domain is by Kimura *et al.* (1984). In this paper, they examined fixed and periodic solutions of N point vortices in a circular boundary. Kimura and Hasimoto (1986) discussed the behavior of two vortices in a semi-circular domain. They investigated the motion of one positive and one negative vortex by perturbing their initial condition which was a fixed point. It is possible to find the configurations of standing vortices (*i.e.*, fixed points, these will be explained in Chapter 5) in Aref's paper (1986). Kimura and Hasimoto (1986) also investigated a billiard type chaos. In their work, Kimura *et al.* (1984) used trajectories and power spectrums to examine these configurations. Another work on semi-circular domain is by Okamoto and Kimura (1989). This work concentrates on one vortex and an advected particle in semi-circular domain with trajectories and Poincaré maps.

Aref (1990) reviewed the chaotic advection when the Eulerian flow field is very simple. A recent work on chaotic advection is by Lupini and Siboni (1991). In their paper, the authors discussed a vortex and an advected particle in an elliptic domain with Poincaré maps.

Eckhardt and Aref (1988) discussed collision dynamics of one and two vortex pairs by reducing the Hamiltonian as in Aref and Pomphrey's work (1982).

In 1987, the IUTAM Symposium 'Fundamental Aspects of Vortex Motion' was held in Tokyo, Japan. Aref and Kambe (1988) summarized this symposium in their paper. Eckhardt (1988) investigated integrable four vortex motion in his work. He searched for the cases where total strength was zero. Aref *et al.* (1989), in their paper gave a general discussion of Lagrangean and Euler representation and explained why Lagrangean point of view is preferred when chaotic motions are investigated. They also investigated the role of dimensionality and vorticity in chaotic motions. Aref *et al.* gave an ABC flow, and a system with two alternatingly opened sinks as an example to the role of vorticity. Pair-up of four vortices and fractal structures are also investigated in this paper. Ünal and Güngör (1991) investigated two point vortices under the perturbation of time by Melnikov method. Interaction of point vortex with solid bodies are investigated by Kaykayoğlu and Ertemer (1987) with computer experiments.

Weiss and McWilliams (1991) discussed whether point vortex motion is ergodic or not. In their investigation they used six point vortices in doubly periodic square domain. Their time and ensemble averages are compared and they found it to be nonergodic.

It has to be noted that none of the papers cited above calculated the Lyapunov exponents (see. Chapter 5.4). Therefore, there is no strong evidence of chaos in low dimensional point vortex system.

1.3. Present Approach

In the present approach vortices in unbounded and bounded domain are discussed. Their fixed points and characteristics of the fixed points are given. Symmetries in these cases and their small perturbations are also investigated. In both unbounded and bounded domain, the minimum number of vortices in order to create chaos are discussed. To show chaotic motion trajectories, power spectrum and Poincaré maps are given. Calculation of Lyapunov exponents are discussed for point vortex motion. Future works on this subject are suggested.

In Chapter 2, formulation and a short discussion about vortex dynamics are given. It is shown that the system is Hamiltonian.

In Chapter 3, dynamical system analysis used in here is discussed. Poincaré maps and Lyapunov exponents are introduced in this Chapter. Chaotic behavior of dynamical systems also given shortly.

Chapter 4 involves a new programming technique: 'Object Oriented Programming.' Application of this technique to numerical analysis and the programs used in this work are introduced.

Results of the computer experiments are given in Chapter 5. Unbounded, circular and semi-circular cases are given separately. Trajectories and power spectrums are drawn for all these cases and Poincaré maps are given for some of the cases.

Chapter 6 shortly discusses the application of statistical mechanics to the point vortex and tries to find an answer to the question 'Could statistical mechanics be used in chaotic systems with small degrees of freedom?

CHAPTER 2
FORMULATION OF THE POINT VORTEX DYNAMICS

A rectilinear vortex is a vortex with a cylindrical vortex tube whose generators are perpendicular to the plane of motion. A point rectilinear vortex (will be mentioned as point vortex for short) has zero diameter (Milne-Thomson, 1958). In this section formulation of unbounded and bounded point vortex motion and its advected particle is given.

2.1. Unbounded Motion

In this work we assume that the fluid is continuous, Newtonian, inviscid, and incompressible. Now governing equations will be derived. This derivation can be found in Şuhubi (1968).

For ideal fluids the equations of motion are

$$\frac{\partial V}{\partial t} + V \nabla V = f - \frac{1}{\rho} \nabla p \tag{2.1}$$

where V is velocity field, f is mass force, and p is pressure and the continuity equation is

$$\frac{\partial \rho}{\partial t} + \nabla (\rho V) = 0 \tag{2.2}$$

where ρ is the density.

For the incompressible flow continuity equation becomes

$$\nabla \cdot V = 0 \tag{2.3}$$

and for the two dimensional case,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2.4}$$

whew u and v are velocities in x and y directions, respectively.

If a scalar function $\psi = \psi(x,y,t)$ is introduced with following conditions

$$u = \frac{\partial \psi}{\partial y}, \qquad v = -\frac{\partial \psi}{\partial x} \tag{2.5}$$

then ψ satisfies Eq. (2.4). This ψ function is called the 'stream function' and it gives stream lines for ψ (x,y,t) = constant.

If the vorticity vector is introduced in two dimensions as,

$$\omega_x = \omega_y = 0 \qquad \omega_z = \omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{2.6}$$

then by substituting ψ into this equation following equation can be obtained,

$$\Delta \psi = -\omega \tag{2.7}$$

From Eq. (2.1) with reducing the right hand side to zero one in the case of no body force, and in the absence of pressure one can write the equation of motion in terms of stream function,

$$\frac{\partial}{\partial t}\Delta\psi + \frac{\partial \psi}{\partial y}\frac{\partial}{\partial x}\Delta\psi - \frac{\partial \psi}{\partial x}\frac{\partial}{\partial y}\Delta\psi = 0 \tag{2.8}$$

By inserting Eq. (2.7)

$$\frac{\partial \omega}{\partial t} + u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = 0 \tag{2.9}$$

can be obtained.

It is possible to model this system with N point vortices by applying a solution to Eq. (2.9) of following form

$$\omega = \sum_{i-1}^{N} \kappa_i \delta(x - x_i(t)) \delta(y - y(t)) \tag{2.10}$$

where δ is Dirac function, κ_α are strengths of point vortices (Aref, 1983). If ψ solved by using Eq. (2.7)

$$\psi = -\frac{1}{4\pi} \sum_{i-1}^{N} \sum_{j-1; j \neq i}^{N} \kappa_i \kappa_j \ln |z_i - z_j| \tag{2.11}$$

can be found where z is the position of a vortex in complex plane. Using Eq.(2.5) the equations modeling the interaction of N point vortices in an incompressible, ideal fluid can be given with

$$\dot{z}_j = -\frac{1}{2\pi} \sum_{i-1}^{N}{}' \frac{\kappa_i}{\bar{z}_j - \bar{z}_i}, \qquad j = 1, \ldots, N \tag{2.12}$$

where κ_i are strengths, the prime over the summation means singular terms $i=j$ are dropped and the bar over z denotes complex conjugate.

It can be seen from Eq.(2.5) that this is a Hamilton system

$$\kappa_j \dot{x}_j = \frac{\partial H}{\partial y_j}, \qquad \kappa_j \dot{y}_j = -\frac{\partial H}{\partial x_j} \tag{2.13}$$

where

$$H = -\frac{1}{4\pi} \sum_{i,j-1}^{N}{}' \kappa_i \kappa_j \ln |z_j - z_i| \tag{2.14}$$

has been called 'kinetic energy interaction' (Aref, 1983).

This conclusion shows that it is possible to model an inviscid, incompressible and two dimensional Newtonian flow with N point vortices.

If Poisson bracket notation is introduced

$$[f,g] = \sum_{j=1}^{N} \frac{1}{\kappa_j}\left(\frac{\partial f}{\partial x_j}\frac{\partial g}{\partial y_j} - \frac{\partial f}{\partial y_j}\frac{\partial g}{\partial x_j}\right) \tag{2.15}$$

then Eq. (2.2) can be written as

$$\dot{x}_j = [x_j,\ H], \qquad \dot{y}_j = [y_j,\ H] \tag{2.16}$$

so there is three well-known integrals of the motion, apart from H itself. These are

$$Q = \sum_{j=1}^{N} \kappa_j x_j, \tag{2.17}$$

$$P = \sum_{j=1}^{N} \kappa_j x_j, \tag{2.18}$$

and

$$I = \sum_{j=1}^{N} \kappa_j (x_j^2 + y_j^2), \tag{2.19}$$

which arise from the invariance of H to translation and rotation of coordinates (Aref, 1983).

2.2. Bounded Motion

Hamiltonian of the unbounded case is mapped using Riemanian mapping (Carrier *et al.*, 1966). Placing image vortices resembles the resulting equations. For the circular bounded case one has to put image vortices of the opposite sign to represent the boundary. Images are placed to $a^2/(x+iy)$ (Fig. 2.1). For a semicircular motion again

image vortices of opposite sign are placed to (x-iy) (Fig. 2.2) if the center of the semi-circle is at the origin and placed above x axis (Kimura *et al*, 1984, Kimura and Hasimoto, 1986).

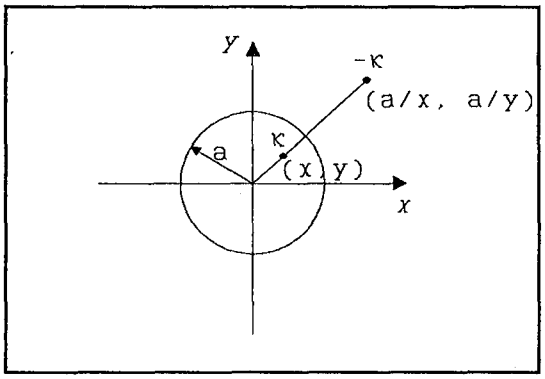


Fig. 2.1

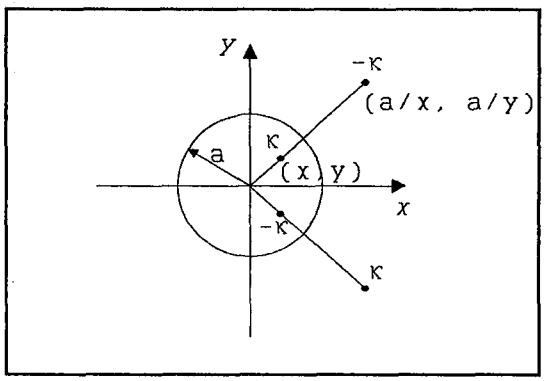


Fig. 2.2

2.3. Advected Particle

If one of the vortices strength is taken zero that vortex will not appear in Hamiltonian and it can be thought as an advected particle. The governing equation will be

$$\dot{\zeta} = -\frac{1}{2\pi} \sum_{i=1}^{N} \frac{\kappa_i}{\zeta - z_i} \qquad \textbf{(2.20)}$$

where ζ is the position of advected particle in complex plane.

This problem was called 'restricted vortex problem' by Aref and Pomphrey (1980) in analogy with the terminology used in celestial mechanics.

CHAPTER 3

POINT VORTEX MOTION AS A HAMILTONIAN DYNAMICAL SYSTEM

It is sufficient to regard a differential equation as a system (Thomson and Stewart, 1986)

$$\dot{x} = f(x) \tag{3.1}$$

where the dot over x means d/dt. Here x = x(t) is a vector valued function of time and f is a smooth function. Vector field f generates a flow ϕ_t that satisfies the Eq.(3.1).

$$\frac{d}{dt}(\phi(x,t))\Big|_{t=\tau} = f(\phi(x,\tau)) \tag{3.2}$$

where t and $\tau \in$ an interval I=(a,b). If vector field does not depend on time explicitly then the system is called autonomous. This will be the case to be analyzed. Non-autonomous systems whose vector fields depend on time will not appear in this work.

The flow $\phi_t(x)$ is a set of solutions of Eq.(3.1). For a given initial condition $x(0)=x_0$ the problem becomes to find the particular solution $\phi(x_0,t)$ with the condition

$$\phi(x_0,0) = x_0. \tag{3.3}$$

When a solution $\phi(x_0,t)$ is obtained analytically or by numerical integration the first way to understand it, would be to display its trajectories on phase space. This can give little information about the behavior of the solution. Power spectrum of one of the trajectories can display systems periodicity characteristics. Another way of understanding the solution is to draw Poincaré map which is explained shortly.

3.1. Poincaré Map

If the systems dimension is greater then displaying trajectories will not give enough information. To overcome this problem its Poincaré map could be used. Poincaré maps allow to reduce the dimension of the system by one. Repeating this method several times it is possible to obtain a 2 dimensional view of the solution (Guckenheimer and Holmes, 1983, Parker and Chua, 1989).

If the dimension of the system is n, a hyperplane Σ of n-1 dimension is introduced. Every crossing of trajectory through this hyperplane is collected and called the Poincaré map (Fig. 3.1).

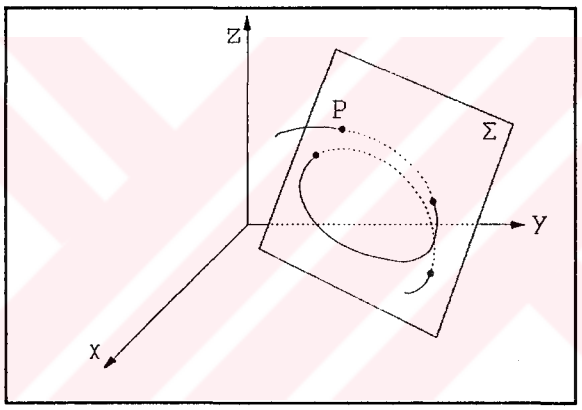


Fig. 3.1

Selecting a hyperplane is an important subject. If it is selected out of limit set then, trajectory will not intersect the hyperplane and no points will be obtained.

3.2. Chaotic Behavior of Dynamical Systems

There is no widely accepted definition of chaos in dynamical systems. This term is used when the solution of a dynamical system is not an equilibrium solution, periodic solution or a quasi-periodic solution.

The meaning of chaos according to Webster dictionary is: "That confusion or confused mass out of which the universe was created; a confused mixture of parts or

elements; a scene of extreme confusion; disorder." These definitions -even "disorder"- does not apply fully the chaos in dynamical systems.

In systems found in nature it is very easy to see and to expect chaos. After making assumptions and mathematically modelling them we can still expect to observe chaotic behavior. But chaos could be seen in one dimensional non-autonomous systems. Even in discrete time systems chaos could be observed. A good example for this is random number generators. As their name implies they are chaotic.

The main features of chaos is loss of information and sensitive dependence to initial conditions. A very small perturbation given to the initial condition generates two very different solutions. In this case it can be said that the system does not remember the initial condition and therefore, perturbations of the initial condition.

In order to understand whether the system is chaotic or not, some criteria will be introduced:

i) time series of the system looks erratic,
ii) the power spectrum exhibits broadband noise,
iii) the autocorrelation function decays rapidly,
iv) the Poincaré map shows space filling points,
v) one of the Lyapunov exponents is positive.

Time series, power spectrum and the Poincaré maps used in this work to investigate chaotic behavior. Lyapunov exponents are explained in next sub-section.

To introduce chaos the Henon-Heiles oscillator example will be given. This system is chosen because its Hamiltonian like point vortex motion. Henon-Heiles oscillator is given by the following equation

$$H(q_1,q_2,p_1,p_2) = \frac{1}{2}p_1^2+\frac{1}{2}p_2^2+\frac{1}{2}q_1^2+q_2^2+q_1^2q_2-\frac{1}{3}q_2^3 = E = Const. \quad \textbf{(3.4)}$$

This system has an so called escape energy of $E=1/6$. Below this value phase space of the system is bounded. It is observed that system is periodic at $E=1/12$ and becomes chaotic as E goes to 1/6.

In Fig. (3.2) a Poincaré section for E=1/12 is shown. It is clear from the figure that at this energy level this system is periodic. Periodicity of a system can be checked from power spectra also. While single peaks shows periodicity, continues ones with disordered peaks means chaotic motion. Fig. (3.3) shows the Poincaré map at E=1/6. Nearly all the area shows mixed flow but there are some islands where no points appears. If the Poincaré map is full of disordered points this shows chaos and mixing. But an ordered Poincaré map does not mean integrable motion for all the cases. In these cases one should look for a positive Lyapunov exponent. Even with small islands we can expect ergodicity is true in a system with a Poincaré map like Fig. (3.3). Although (von Alberti, 1990) showed that Henon-Heiles system is almost ergodic at E=1/6.

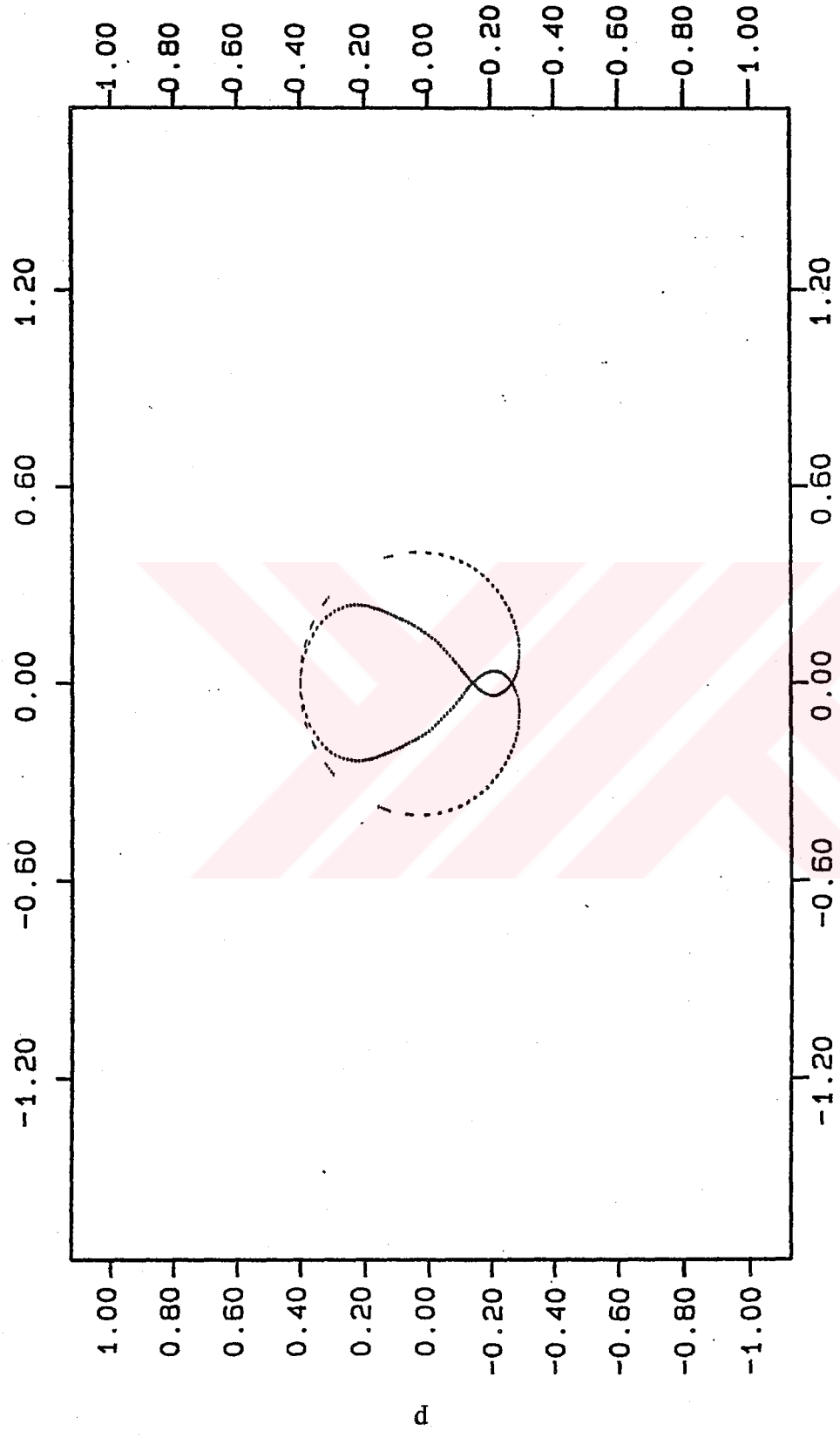


Fig. (3.2) Poincaré sections of Henon-Heiles E=1/12

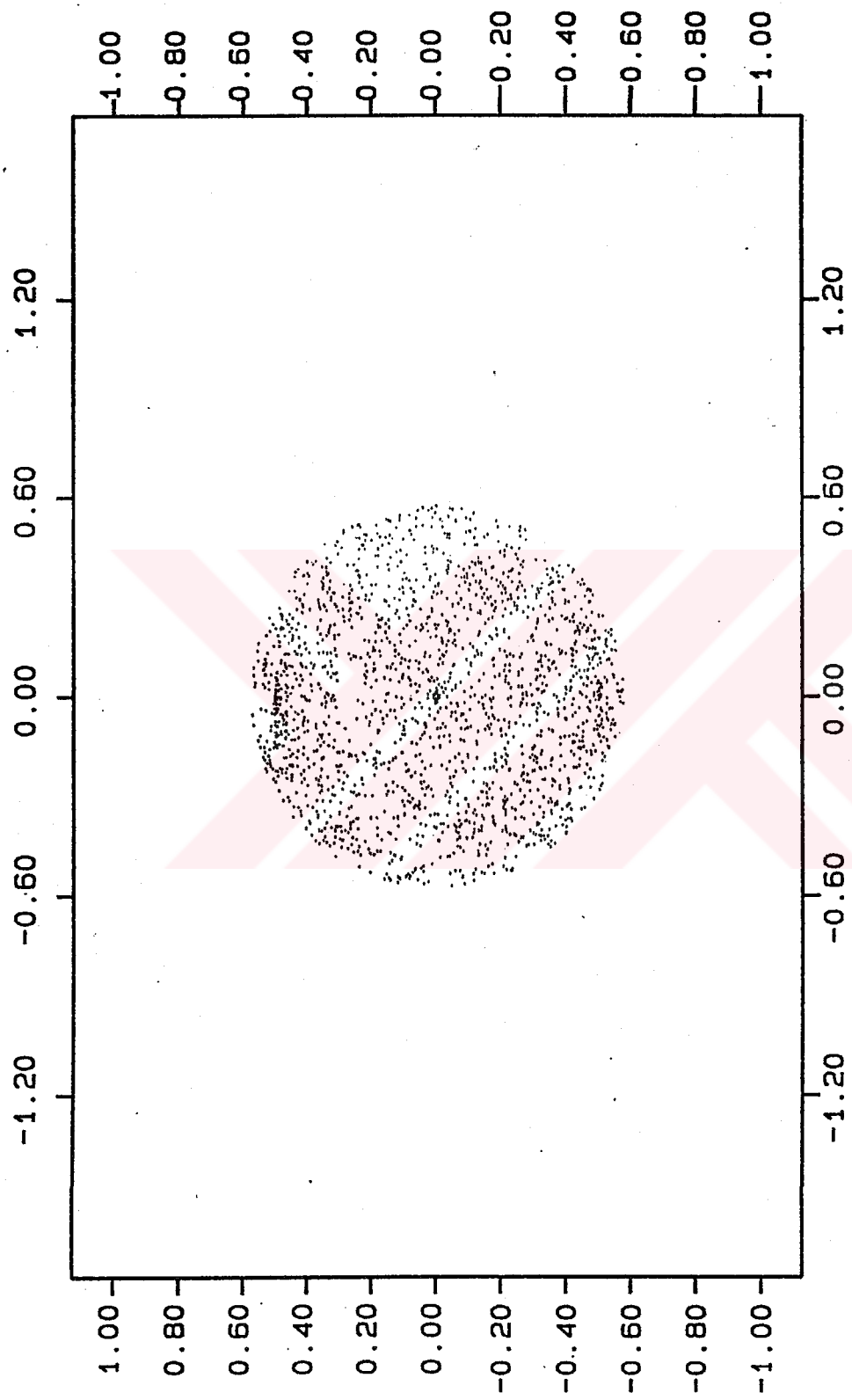


Fig. (3.3) Poincaré sections of Henon-Heiles E=1/6

3.3. Stability and Lyapunov Exponents

Any system containing at least one positive Lyapunov exponent is defined to be chaotic. The magnitude of the exponent reflecting the time scale on which system dynamics become unpredictable (Wolf *et al.*, 1985).

Lyapunov exponents are the most important proof (Goldhirsch *et al.*, 1987, Wolf *et al.*, 1985) of chaos as understood from the above statement.

Lyapunov exponents are the average exponential rates of divergence or convergence of nearby orbits in phase space (Wolf *et al.*, 1985). If we assume that there is a sphere of infinitesimal radius and observe its shape along the trajectory it will deform to ellipsoid. Mathematically Lyapunov exponents are defined as follows:

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \ln\frac{p_i(t)}{p_i(0)}, \qquad i = 1,\ldots,n \tag{3.5}$$

where $p_i(t)$ is the length of the ellipsoidal principal axis.

Any continues time-dependent dynamical system without a fixed point will have at least one zero exponent and any dissipative dynamical system will have at least one negative Lyapunov exponent. Therefore, for a 4 dimensional chaotic system the only possibilities are (+,0,-,-), (+,0,0,-), (+,+,0,-).

In order to find the Lyapunov exponents we should introduce

$$\begin{aligned} \dot{\Phi} &= J\,\Phi \\ J_{i,j} &= \frac{\partial f_i}{\partial x_j} \end{aligned} \tag{3.6}$$

where J is the Jacobean matrix. p_i are eigenvalues of Φ.

For n point vortices Jacobian matrix is as follows:

$$\frac{\partial \dot{x}_j}{\partial x_i}\Big|_{i=j} = \frac{1}{\pi} \sum_{k=1}^{N}{}' \kappa_k \frac{(y_j-y_k)(x_j-x_k)}{[(x_j-x_k)^2+(y_j-y_k)^2)]^2} \tag{3.7}$$

$$\frac{\partial \dot{x}_j}{\partial x_i}\Big|_{i\neq j} = -\frac{\kappa_i}{\pi} \frac{(y_j-y_i)(x_j-x_i)}{[(x_j-x_i)^2+(y_j-y_i)^2]^2} \tag{3.8}$$

$$\frac{\partial \dot{x}_j}{\partial y_i}\Big|_{i=j} = -\frac{1}{2\pi} \sum_{k=1}^{N}{}' \kappa_k \frac{(x_j-x_k)^2-(y_j-y_k)^2}{[(x_j-x_k)^2+(y_j-y_k)^2]^2} \tag{3.9}$$

$$\frac{\partial \dot{x}_j}{\partial y_i}\Big|_{i\neq j} = \frac{\kappa_i}{2\pi} \frac{(x_j-x_i)^2-(y_j-y_i)^2}{[(x_j-x_i)^2+(y_j-y_i)^2]^2} \tag{3.10}$$

$$\frac{\partial \dot{y}_j}{\partial x_i}\Big|_{i=j} = -\frac{1}{2\pi} \sum_{k=1}^{N}{}' \kappa_k \frac{(x_j-x_k)^2-(y_j-y_k)^2}{[(x_j-x_k)^2+(y_j-y_k)^2]^2} \tag{3.11}$$

$$\frac{\partial \dot{y}_j}{\partial x_i}\Big|_{i\neq j} = \frac{\kappa_i}{2\pi} \frac{(x_j-x_i)^2-(y_j-y_i)^2}{[(x_j-x_i)^2+(y_j-y_i)^2]^2} \tag{3.12}$$

$$\frac{\partial \dot{y}_j}{\partial y_i}\Big|_{i=j} = -\frac{1}{\pi} \sum_{k=1}^{N}{}' \kappa_k \frac{(x_j-x_k)(y_j-y_k)}{[(x_j-x_k)^2+(y_j-y_k)^2]^2} \tag{3.13}$$

$$\frac{\partial \dot{y}_j}{\partial y_i}\Big|_{i\neq j} = \frac{\kappa_i}{\pi} \frac{(x_j-x_i)(y_j-y_i)}{[(x_j-x_i)^2+(y_j-y_i)^2]^2} \tag{3.14}$$

These equations forms Jacobian matrix as

where i=1,...,n; j=1,...,n and J is a 2*n×2*n matrix. The symmetries of Jacobian matrix for point vortex is as follows:

$$J(i,\ j) = \frac{\partial \dot{x}_i}{\partial x_j}, \qquad J(i,\ j+n) = \frac{\partial \dot{x}_i}{\partial y_j}$$
$$J(i+n,\ j) = \frac{\partial \dot{y}_i}{\partial x_j}, \qquad J(i+n,\ j+n) = \frac{\partial \dot{y}_i}{\partial y_j} \tag{3.15}$$

$$\begin{aligned} &J(i,\ i) = -J(i+n,\ j+n) \\ &J(i,\ i+n) = J(i+n,\ i) \\ &J(i,\ j) = J(j,i) = -J(i+n,\ j+n) = -J(j+n,\ i+n) \\ &J(i,\ j+n) = J(j,i+n) = J(j+n,\ i) = J(i+n,\ j) \end{aligned} \tag{3.16}$$

Then a 4×4 matrix would be as follows

$$J = \begin{bmatrix} A & B & C & D \\ B & E & D & C \\ C & D & -A & -B \\ D & C & -B & -E \end{bmatrix} \tag{3.17}$$

This symmetries are used to calculate Jacobian matrix easier. Exploiting this form of the Jacobian matrix is an ongoing process.

CHAPTER 4
A NUMERICAL APPROACH FOR DYNAMICAL SYSTEM TOOLS

The development of new ideas in the field of nonlinear dynamics was greatly assisted by the invention of computers. They are indispensable even for the cases where formal results are quite readily obtainable (Sagdeev *et al.*, 1988). In this chapter numerical methods to calculate Lyapunov exponents, power spectra and Poincaré map are discussed. A new programming technique is introduced.

4.1. Object Oriented Programming

When the first computers were invented, programs were written by Machine Code which only contains numbers as commands. Machine Code programming is very difficult to code because it only consists of numbers and numbers does not mean much to people. When the required software became complicated assembler was invented. Assembler is very similar to Machine Code but instead of numbers there are mnemonics in its command set. It was easier to remember the commands but difficulties arose when the software became complicated. Then first high level languages were developed. They provided more tools to handle the problems. In 1960's structured languages like C and Pascal came into being. With the help of structured languages, writing complex programs became simpler (Schildt, 1991).

All of the programming languages mentioned above were the outcomes of Processes Oriented Programming (POP). The importance in POP is given to the task. And then came the Object Oriented Programming (OOP). OOP gives the priority to objects defined not the task to be performed. In OOP, one teaches objects what they will do, and when needed just tell them to do the task.

OOP is not a programming language. It does not even require an Object Oriented Language (OOL). It is possible to do OOP with Assembler, Pascal or C. OOP is a way of thinking in program design.

Doing OOP effectively requires an OOL. Any OOL has minimum requirements of encapsulation of data, inheritance, polymorphism and objects. An object is a logical entity that contains both data and code in order to manipulate the data. In other words, objects know what they are (variables) and how they behave (functions). With the encapsulation of data, it is possible to restrict other objects or program parts to reach a member of the object. Polymorphism means one interface, multiple methods. With polymorphism, one can give the same name to similar functions. Another useful feature is inheritance. With this feature it is possible for objects to inherit each others characteristics.

The main advantage of OOP is reusability. Once an object is created, it can be used in any program without any change. Existing objects can be modified very simply without changing the original object with the help of inheritance. Designing programs is easier because each object is self-sufficient and it can be independently designed. Once the fundamental objects are created the programming becomes only to tell the objects what to do. This is a big difference because in POP, at every step one has to tell the variables (objects) what they should do and how to do it. Since objects know how to perform operations, in OOP one just tells them what to do.

4.2. Using OOP in Numerical Analysis

Numerical analysis consists of several objects and methods to manipulate those objects. Some objects of numerical analysis are scalar numbers, sets, vectors, matrices, polygons, nonlinear equations etc. while multiplication, orthogonalization, numerical integration are examples of methods.

Defining frequently used objects such as vectors, would save considerable amount of production time. Nowadays, it is possible to buy this kind of objects from market. This kind of libraries are more creative then large already-compiled programs.

4.3. Objects Created for the Programs

The C programming language with its flexibility, speed and most important of all portability around operating systems is one of the most widely accepted languages. As a OOL C++ is used. C++ could be defined as C with objects. It has the features of C with the added Object Oriented capabilities. When the objects are given, a C can easily use them in his/her programs.

For the programs required in dynamical system analysis following objects are created: sets, vectors, matrix, hyperplane and ODE.

Set object knows its elements and the number of elements it contains. It knows how to allocate and free memory, how to return and set required element and some mathematical operations. Vector object is inherits all the features of sets. And vector has some special methods of its own. Some of these methods are scalar multiplication and normalization. Matrix object is inherited from sets too with special methods like orthogonalization. Also there are some methods that link vector and matrix objects such as multiplication of a matrix with a vector. Again hyperplane is inherited from set with special methods such as giving the distance of a point to its surface.

ODE object is for defining ordinary differential equations. It knows the equations and the dimension of the system. From this object, new objects are inherited for each new system. This allows two or more equations to use the same functions (e.g. numerical integrator) in the same program.

With the help of these objects a C statement like this

```
double x[4], y[4], z[4];
```

```
for (i=1; i<=4, i++)  x[i] = y[i]*z[i];
```

becomes

```
vector x(4), y(4), z(4);
x = y*z;
```

in C++. When more complex statements will be used, C++ will be more easy to read. In table (4.1) Runge-Kutta subroutine used in programs is compared with its C version.

Table 4.1
Comparison of C++ with C

```
void rk4(vector &y,vector &dydx, double x,
double h, vector &yout, ODE &eq)
{
  double xh,hh,h6;

  vector dym(eq.getdim()), dyt(eq.getdim()),
         yt(eq.getdim());
  hh=h*0.5;
  h6=h/6.0;
  xh=x+hh;
  yt=y+hh*dydx;
  eq.derivs(xh,yt,dyt);
  yt=y+hh*dyt;
  eq.derivs(xh,yt,dym);
  yt=y+h*dym;
  dym=dym+dyt;
  eq.derivs(x+h,yt,dyt);
  yout=y+h6*(dydx+dyt+2.0*dym);
}
```

```
void rk4(y,dydx,n,x,h,yout,derivs)
float y[],dydx[],x,h,yout[];
void (*derivs)();
int n;
{
  int i;
  float xh,hh,h6,*dym,*dyt,*yt,*vector();
  void free_vector();

  dym=vector(1,n);
  dyt=vector(1,n);
  yt=vector(1,n);
  hh=h*0.5;
  h6=h/6.0;
  xh=x+hh;
  for (i=1;i<=n;i++)
    yt[i]=y[i]+hh*dydx[i];
  (*derivs)(xh,yt,dyt);
  for (i=1;i<=n;i++)
    yt[i]=y[i]+hh*dyt[i];
  (*derivs)(xh,yt,dym);
  for (i=1;i<=n;i++) {
    yt[i]=y[i]+h*dym[i];
    dym[i] += dyt[i];
  }
  (*derivs)(x+h,yt,dyt);
  for (i=1;i<=n;i++)
    yout[i]=y[i]+h6*
    (dydx[i]+dyt[i]+2.0*dym[i]);
  free_vector(yt,1,n);
  free_vector(dyt,1,n);
  free_vector(dym,1,n);
}
```

4.4. Algorithms for Dynamical System Tools

A step size controlled Runge-Kutta algorithm and Bulirch-Stoer algorithm is modified to use with C++ and above explained objects as numerical integrators (Press *et al.*, 1989). The power spectrums drawn by maximum entropy method are also modified (Press *et al.*, 1989).

For the Poincaré mapping program, time-step halving method is used (Parker and Chua, 1989). Hyperplane objects are used in this study to define the hyperplane Σ.

For the Lyapunov exponents program, two methods are discussed (Stoop *et al.*, 1991, Wolf *et al.*, 1985). Of the two methods given by Stoop et al. converges more quickly but since it requires a single value decomposition algorithm it still takes too much time. Because the method given by Wolf *et al.* (1985) is simpler, this method is used. Eckman and Ruelle (1992) discusses the validity of estimating magnitudes of dimensions and Lyapunov exponents for small amount of data points. They suggest that taking 10^6 data points will probably to give a wrong magnitude. Calculating Lyapunov exponents for a 4 vortex system means 20 equations will be integrated at each step. If each step is 0.1 units of time than this takes about 30 seconds for an accuracy of 10^{-6} on an 80286 PC with a 80287 arithmetic co-processor. For 1,000,000 data points, this takes approximately 347 days of computation and it may still have a wrong magnitude.

Our experiments with Lorenz and Rossler-chaos equations, whose Lyapunov exponents were obtained analytically, shows that even when the magnitudes were low, the signs of the exponents were correct. But both Lorenz and Rossler-chaos equations are 3 dimensional and it is not possible to assume that the signs of the exponents will be correct for a small number of data points in a four vortex problem. Therefore, trying to find analytical simplifications and a suitable method for calculation of the positive Lyapunov exponents are ongoing processes.

CHAPTER 5
RESULTS OF DYNAMICAL SYSTEM ANALYSIS FOR THE POINT VORTEX DYNAMICS

Different configurations of point vortices are examined in this Chapter. Their trajectories, power spectrums, and Poincaré sections are displayed.

5.1. Unbounded Domain

In a point vortex system, a fixed point signifies 'standing vortices'. Standing vortices preserve their initial conditions at all times. Stability of these fixed points are investigated by applying small perturbations to the original initial conditions of the standing vortices. If the solution of a system that is close to the fixed point at the beginning remains close at all times, this is said to be a stable fixed point. If solutions converge to a fixed point as $t \mapsto \infty$, this is called asymptotically stable (Wiggins 1990).

Four is the minimum number of point vortices in order to obtain chaotic behavior in an unbounded motion as mentioned above. Three vortex motion is wholly integrable as shown by Gröbli in his 1877 paper (Aref, 1983). Since a Hamilton system is integrable if it has N integrals in involution, it is guaranteed that the motion of three vortices is integrable for any values of the vortex strengths (Aref and Pomphrey 1982). These two cases will be presented in two following sub-sections.

5.1.1. Case: $N_* \leq 3$

One point vortex in an unbounded domain is a fixed solution for every initial condition since the left-hand side of the governing ODEs vanishes.

For two point vortices equations of motion (Eq. (2.12)) become

$$\begin{aligned} \dot{x}_1 &= -\frac{1}{2\pi}\frac{y_1-y_2}{(x_1-x_2)^2+(y_1-y_2)^2}\,, \\ \dot{y}_1 &= \frac{1}{2\pi}\frac{x_1-x_2}{(x_1-x_2)^2+(y_1-y_2)^2}\,, \\ \dot{x}_2 &= -\dot{x}_1\,, \\ \dot{y}_2 &= -\dot{y}_1\,. \end{aligned} \tag{5.1}$$

For a fixed point to exist in this case, y_1-$y_2 = 0$ and x_1-$x_2 = 0$ must be satisfied. The only possibility is $y_1=y_2$ and $x_1=x_2$. But this configuration makes the Hamiltonian infinite. Therefore, there is no fixed point for two vortices in unbounded domain.

For three vortices, a fixed point for the initial configuration shown in Fig. (5.1) is found.

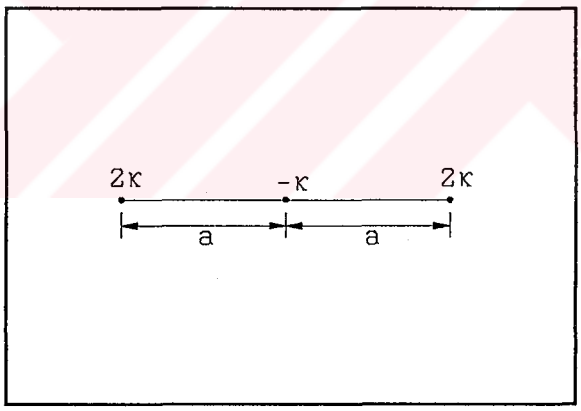


Fig. 5.1

A small perturbation in the middle point vortex shows that this is not a stable fixed point Fig. (5.2).

5.1.2. Case: N. > 3

A fixed solution for four point vortices is given by Aref, 1986 for the following configuration (Fig. 5.3).

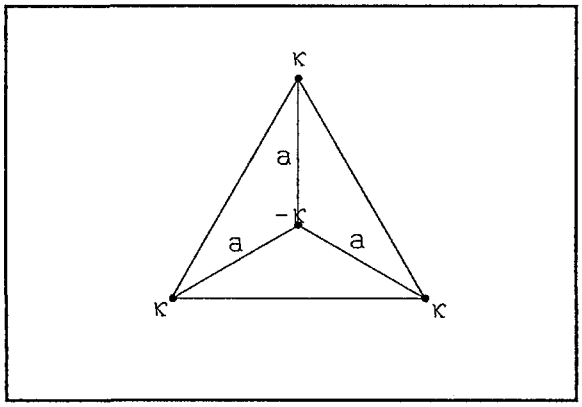


Fig. 5.3

There are fixed points for higher numbers of point vortices. For example, for five vortices a square with $+\kappa$ vortices in corners and a -1.5κ vortex in the center gives a fixed solution. Perturbation of this initial condition shows that this is not a stable fixed point (Fig. 5.6).

There are configurations that give fixed solutions for larger numbers of point vortices (Aref 1986).

Symmetries of configuration in four point vortices may result in integrable motion as shown in Fig. (5.7). In this configuration, vortices of identical strength placed at the corners of a rectangle. This configuration yields a quasi-periodic motion. Power spectrum shows its periodicity (Fig. 5.8). A small perturbation to this configuration gives Fig. (5.9). But there is no indication of chaos when the power spectrum is investigated (Fig. (5.10)). If the perturbation is large, then the trajectories (Fig. (5.11)) and the power spectrum (Fig. (5.12)) show that it is a chaotic motion. This result can be explained with KAM (Kolmogorov, Arnold, and Moser) theorem. The theorem states that if among other conditions, perturbation is sufficiently small, trajectories will continue to lie on smooth N-dimensional integral surfaces (Ford, 1974). When perturbation is large, then trajectories no longer stay on the tori.

If the shape of the rectangle is preserved but, the strength of one of the vortices is changed then this gives a chaotic behavior (fig (5.13)). Power spectrum of this

configuration (Fig. (5.14)) shows the noise which identifies chaos.

Another symmetry yielding integrable motion is a face-centered equilateral triangle. As it was discussed above taking the strength of the center vortex of opposite sign gives a fixed solution. If all vortices are of identical strengths then, the vortices at the corners follows circular trajectories while the vortex at the center is stationery.

Fig. (5.15) shows the trajectories when a small perturbation is introduced to the position of the vortex at the center. Power spectrum (Fig. (5.16)) shows noise and Poincaré map (Fig. (5.17)) shows space filling points. All of this results identify chaos.

Symmetry of the line configuration can yield integrable motion. This configuration with identical vortices are discussed by Aref and Pomphrey (1982). Fig. (5.18) shows the trajectory of a configuration where the distances between outside and inside vortices are half of the distance between the middle vortices. Its power spectrum (Fig. (5.19)) and trajectory show that it is a regular motion.

Same initial conditions with two +1 and two -5 vortices are discussed in four possible configurations. The first one is the case where the two vortices of -5 strength are placed inside (Fig. (5.20)). This case starts as a quasi-periodic motion and after some time trajectories, begin to go beyond their initial limit set. Its power spectrum (Fig. (5.21)) does not shows noise but the Poincaré map (Fig. (5.22)) shows space filling points and this is an indication of chaos.

If the -5 point vortices are placed outside, trajectory (Fig. (5.23)), power spectrum (Fig. (5.24)), and Poincaré map (Fig. (5.25)) show that this is an integrable motion. All other configurations are also integrable. Fig. (5.26) and Fig. (5.27) are the trajectory and power spectrum of +1, +1, -5, -5, configuration and, Fig. (5.28) and Fig. (5.29) are of +1, -5, +1, -5.

This is an interesting result. Among all possible combinations of strengths only one shows chaotic motion. Further numerical simulations showed that if vortices are placed with equal separations then all four configurations become integrable. This result could not be found in the literature and the investigation of this situation is an ongoing process.

Pairing up of four point vortices to infinity is discussed by Aref *et al.* (1989). Some of the configurations are reproduced here. Fig. (5.30) is a line configuration with equally separated vortices of +1, +1, -1, -1 strength. It displays a regular behavior. If this configuration is perturbed then an asymmetric motion is observed (Fig. (5.31)). As it is seen from the trajectories, two pairs remain close together, but after some time one of the two pairs captures one vortex from the other pair, makes a spin and returns the point vortex to its original partner. Long time behavior of this configuration shows that there is no order in the numbers and the spacings of those spins.

5.2. Bounded Domain

In circular domain, there are fixed points for N/2 positive and N/2 negative point vortices for $N \geq 2$. For N positive vortices there are no fixed points. Fixed points can be obtained by placing equidistant vortices on a circle with a radius of

$$a = [\sqrt{4N^2+1} - 2N]^{1/(2N)} \qquad (5.2)$$

With vortices of the same sign, one can obtain periodic motion if those vortices are placed on a circle with equal separations (Kimura *et al.*, 1984).

In a circular domain, three vortices would be enough to obtain chaotic motion. To show this, three vortices of the same sign are placed on a line above y axis with equal separations. Fig. (5.32) shows trajectories of these configurations and Fig (5.33) gives the power spectrum. Fig. (5.34) shows trajectories of the same configuration where the center vortex is of the opposite sign. Power spectrum (Fig. (5.35)) shows noise and Poincaré map (Fig. (5.36)) shows space filling points. These are indications

of chaos.

Also a billiard type chaos (Sagdeev, *et al.*, 1988) example is given (Fig. (5.37) and Fig. (5.38)) with two negative and two positive point vortices. In this type of behavior, vortices stay near the boundary most of the time. Since positive and negative vortices turn at different directions they sometimes meet on the boundary and they form a pair and travel together until they hit the boundary where they separate again.

In semi-circular domain there is both a stable and an unstable fix point for two vortices. The stable fixed point can be found by using Eq. (5.2) with obtaining N twice the number of point vortices (Kimura and Hasimoto, 1986).

In a semi-circular domain, it is found that two vortices behave chaotic. A perturbation applied to the stable fix point results in a quasi-periodic behavior (Fig. (5.39) and Fig (5.40)). If it is perturbed more (Fig. (5.41), Fig. (5.42), and Fig. (5.43)) then this gives a chaotic motion. Fig. (5.44) shows a billiard type chaos. Its power spectrum (Fig. (5.45)) and Poincaré map (Fig. (5.46)) shows that its chaotic.

First two configurations of semi-circular domain are investigated with two positive vortices. First configuration shows a very complex but quasi-periodic motion (Fig. (5.47)) as its power spectrum proves this result (Fig. (5.48)). Second configuration in this case does not gives chaotic behavior (Fig. (5.49)) as its power spectrum shows (Fig. (5.50)) it is quasi-periodic.

Table 5.1: Initial Conditions

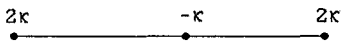
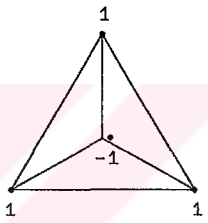
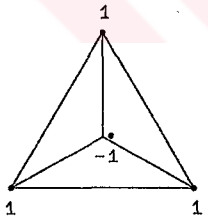
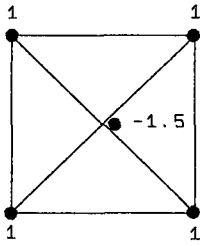
	Initial Conditions	Configuration
Fig (5.2) Page: 39	$x_1=-1$ $y_1=0$ $\kappa_1=2$ $x_2=0.1$ $y_2=0$ $\kappa_2=-1$ $x_3=1$ $y_3=0$ $\kappa_3=2$	
Fig (5.4) Page: 40	$x_1=0.2$ $y_1=0$ $\kappa_1=-1$ $x_2=0$ $y_2=1$ $\kappa_2=1$ $x_3=\cos 60$ $y_3=-0.5$ $\kappa_3=1$ $x_4=\cos 60$ $y_4=-0.5$ $\kappa_4=1$	
Fig. (5.5) Page: 41	$x_1=0.3$ $y_1=0$ $\kappa_1=-1$ $x_2=0$ $y_2=1$ $\kappa_2=1$ $x_3=\cos 60$ $y_3=-0.5$ $\kappa_3=1$ $x_4=\cos 60$ $y_4=-0.5$ $\kappa_4=1$	
Fig. (5.6) Page: 42	$x_1=1$ $y_1=1$ $\kappa_1=1$ $x_2=1$ $y_2=-1$ $\kappa_2=1$ $x_3=-1$ $y_3=1$ $\kappa_3=1$ $x_4=-1$ $y_4=-1$ $\kappa_4=1$ $x_5=0.01$ $y_5=0$ $\kappa_5=-1.5$	

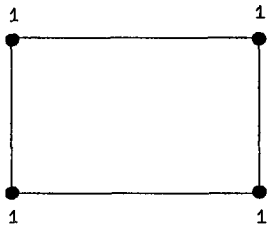
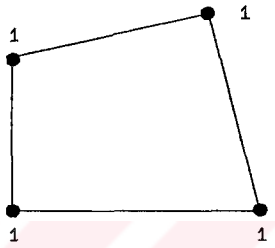
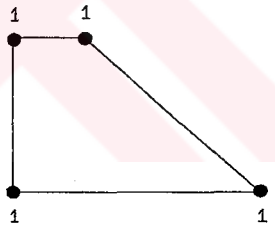
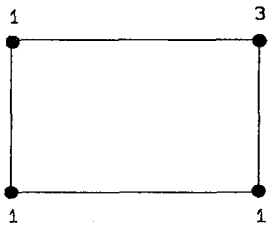
Fig. (5.7) Page: 43	$x_1=1.5$ $y_1=1$ $\kappa_1=1$ $x_2=1.5$ $y_2=-1$ $\kappa_2=1$ $x_3=-1.5$ $y_3=1$ $\kappa_3=1$ $x_4=-1.5$ $y_4=-1$ $\kappa_4=1$	
Fig. (5.9) Page: 45	$x_1=1$ $y_1=1.5$ $\kappa_1=1$ $x_2=1.5$ $y_2=-1$ $\kappa_2=1$ $x_3=-1.5$ $y_3=1$ $\kappa_3=1$ $x_4=-1.5$ $y_4=-1$ $\kappa_4=1$	
Fig. (5.11) Page: 47	$x_1=-0.5$ $y_1=1$ $\kappa_1=1$ $x_2=1.5$ $y_2=-1$ $\kappa_2=1$ $x_3=-1.5$ $y_3=1$ $\kappa_3=1$ $x_4=-1.5$ $y_4=-1$ $\kappa_4=1$	
Fig. (5.13) Page: 49	$x_1=1.5$ $y_1=1$ $\kappa_1=6$ $x_2=1.5$ $y_2=-1$ $\kappa_2=1$ $x_3=-1.5$ $y_3=1$ $\kappa_3=1$ $x_4=-1.5$ $y_4=-1$ $\kappa_4=1$	

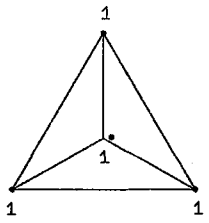
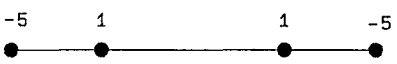
Fig. (5.15) Page: 51	$x_1=0.1 \quad y_1=0 \quad \kappa_1=1$ $x_2=0 \quad y_2=1 \quad \kappa_2=1$ $x_3=\cos 60 \quad y_3=-0.5 \quad \kappa_3=1$ $x_4=\cos 60 \quad y_4=-0.5 \quad \kappa_4=1$	
Fig. (5.18) Page: 54	$x_1=-1 \quad y_1=0 \quad \kappa_1=1$ $x_2=-0.5 \quad y_2=0 \quad \kappa_2=1$ $x_3=0.5 \quad y_3=0 \quad \kappa_3=1$ $x_4=1 \quad y_4=0 \quad \kappa_4=1$	
Fig. (5.20) Page: 56	$x_1=-1 \quad y_1=0 \quad \kappa_1=1$ $x_2=-0.5 \quad y_2=0 \quad \kappa_2=-5$ $x_3=0.5 \quad y_3=0 \quad \kappa_3=-5$ $x_4=1 \quad y_4=0 \quad \kappa_4=1$	
Fig. (5.23) Page: 59	$x_1=-1 \quad y_1=0 \quad \kappa_1=-5$ $x_2=-0.5 \quad y_2=0 \quad \kappa_2=1$ $x_3=0.5 \quad y_3=0 \quad \kappa_3=1$ $x_4=1 \quad y_4=0 \quad \kappa_4=-5$	

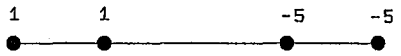
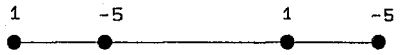
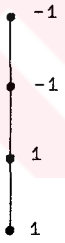
Fig. (5.26) Page: 62	$x_1=-1$ $y_1=0$ $\kappa_1=1$ $x_2=-0.5$ $y_2=0$ $\kappa_2=1$ $x_3=0.5$ $y_3=0$ $\kappa_3=-5$ $x_4=1$ $y_4=0$ $\kappa_4=-5$	
Fig. (5.28) Page: 64	$x_1=-1$ $y_1=0$ $\kappa_1=1$ $x_2=-0.5$ $y_2=0$ $\kappa_2=-5$ $x_3=0.5$ $y_3=0$ $\kappa_3=1$ $x_4=1$ $y_4=0$ $\kappa_4=-5$	
Fig. (5.30) Page: 66	$x_1=0$ $y_1=0$ $\kappa_1=-1$ $x_2=0$ $y_2=0.5$ $\kappa_2=-1$ $x_3=0$ $y_3=1$ $\kappa_3=1$ $x_4=0$ $y_4=1.5$ $\kappa_4=1$	
Fig. (5.31) Page: 67	$x_1=0$ $y_1=-0.1$ $\kappa_1=-1$ $x_2=0$ $y_2=0.5$ $\kappa_2=-1$ $x_3=0$ $y_3=1$ $\kappa_3=1$ $x_4=0$ $y_4=1.5$ $\kappa_4=1$	

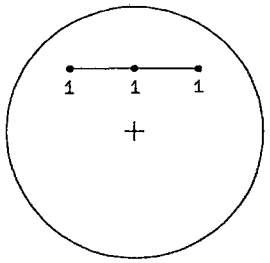
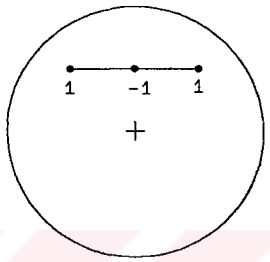
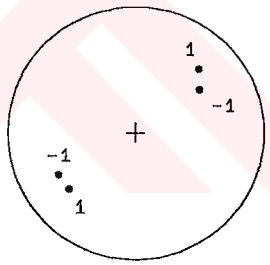
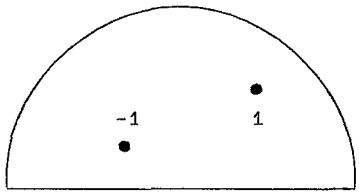
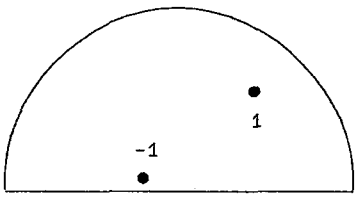
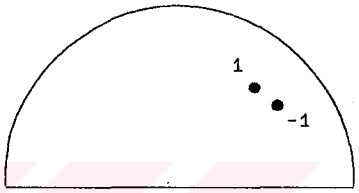
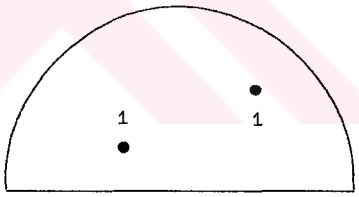
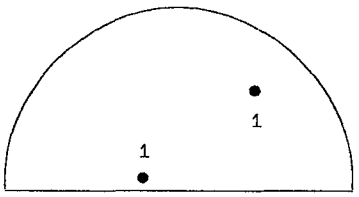
Fig. (5.32) Page: 68	$x_1=-0.5 \quad y_1=0.5 \quad \kappa_1=1$ $x_2=0 \quad y_2=0.5 \quad \kappa_2=1$ $x_3=0.5 \quad y_3=0.5 \quad \kappa_3=1$	
Fig. (5.34) Page: 70	$x_1=-0.5 \quad y_1=0.5 \quad \kappa_1=1$ $x_2=0 \quad y_2=0.5 \quad \kappa_2=-1$ $x_3=0.5 \quad y_3=0.5 \quad \kappa_3=1$	
Fig. (5.37) Page: 72	$x_1=0.499757 \quad y_1=0.499757 \quad \kappa_1=1$ $x_2=0.464808 \quad y_2=0.390020 \quad \kappa_2=-1$ $x_3=-0.499757 \quad y_3=-0.499757 \quad \kappa_3=-1$ $x_4=-0.541412 \quad y_4=-0.541412 \quad \kappa_4=1$	
Fig (5.39) Page: 75	$x_1=0.418846 \quad y_1=0.418846 \quad \kappa_1=1$ $x_2=-0.176778 \quad y_2=0.176778 \quad \kappa_2=-1$	

Fig. (5.41) Page: 77	$x_1=0.418846$ $y_1=0.418846$ $\kappa_1=1$ $x_2=-0.058336$ $y_2=0.058336$ $\kappa_2=-1$	
Fig. (5.44) Page: 80	$x_1=0.418846$ $y_1=0.418846$ $\kappa_1=1$ $x_2=-0.453757$ $y_2=0.380758$ $\kappa_2=-1$	
Fig. (5.47) Page: 83	$x_1=0.418846$ $y_1=0.418846$ $\kappa_1=1$ $x_2=-0.176778$ $y_2=0.176778$ $\kappa_2=1$	
Fig. (5.49) Page: 85	$x_1=0.418846$ $y_1=0.418846$ $\kappa_1=1$ $x_2=-0.058336$ $y_2=0.058336$ $\kappa_2=1$	

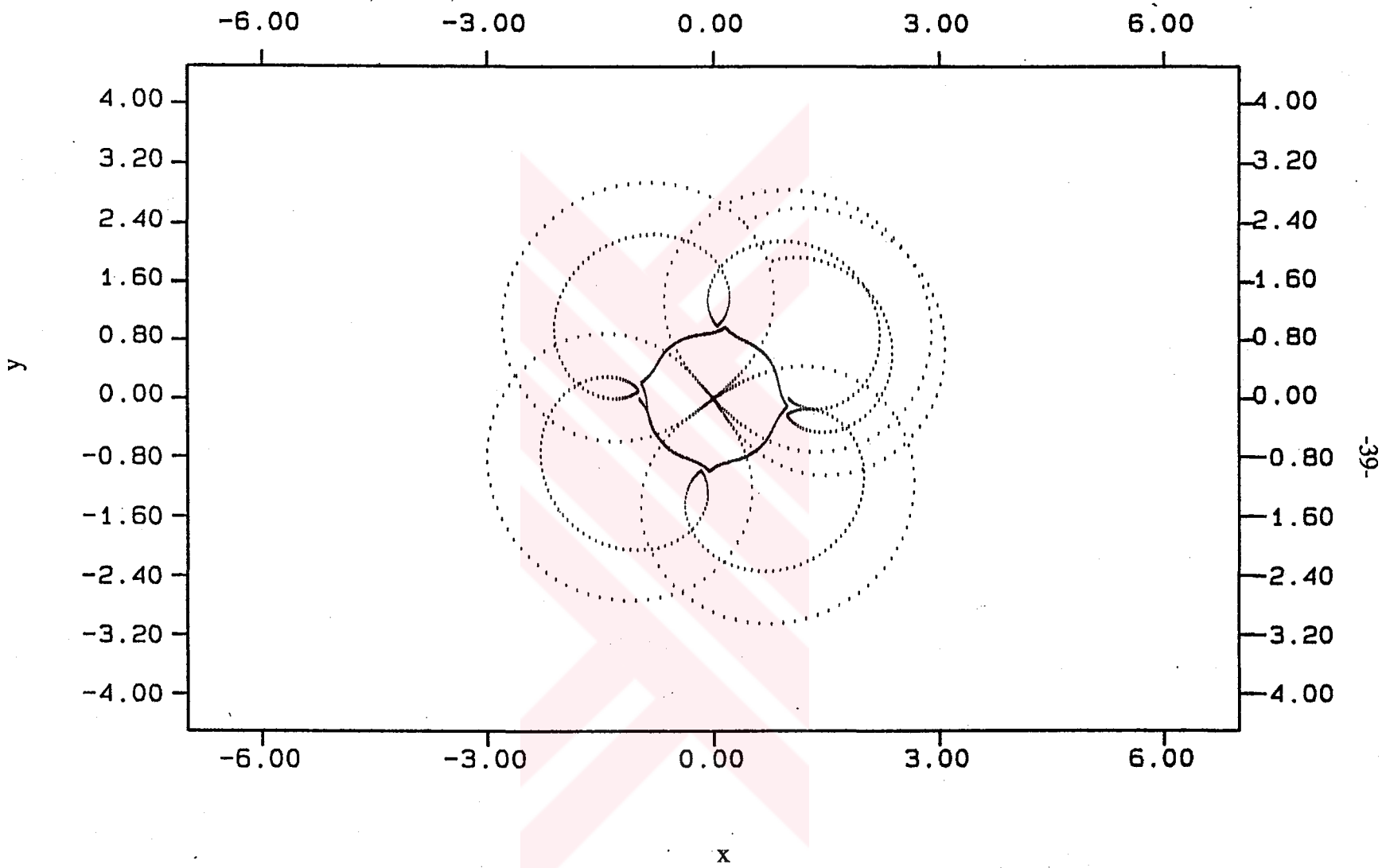


Fig. (5.2) Trajectories of three vortices with small perturbation from fixed points

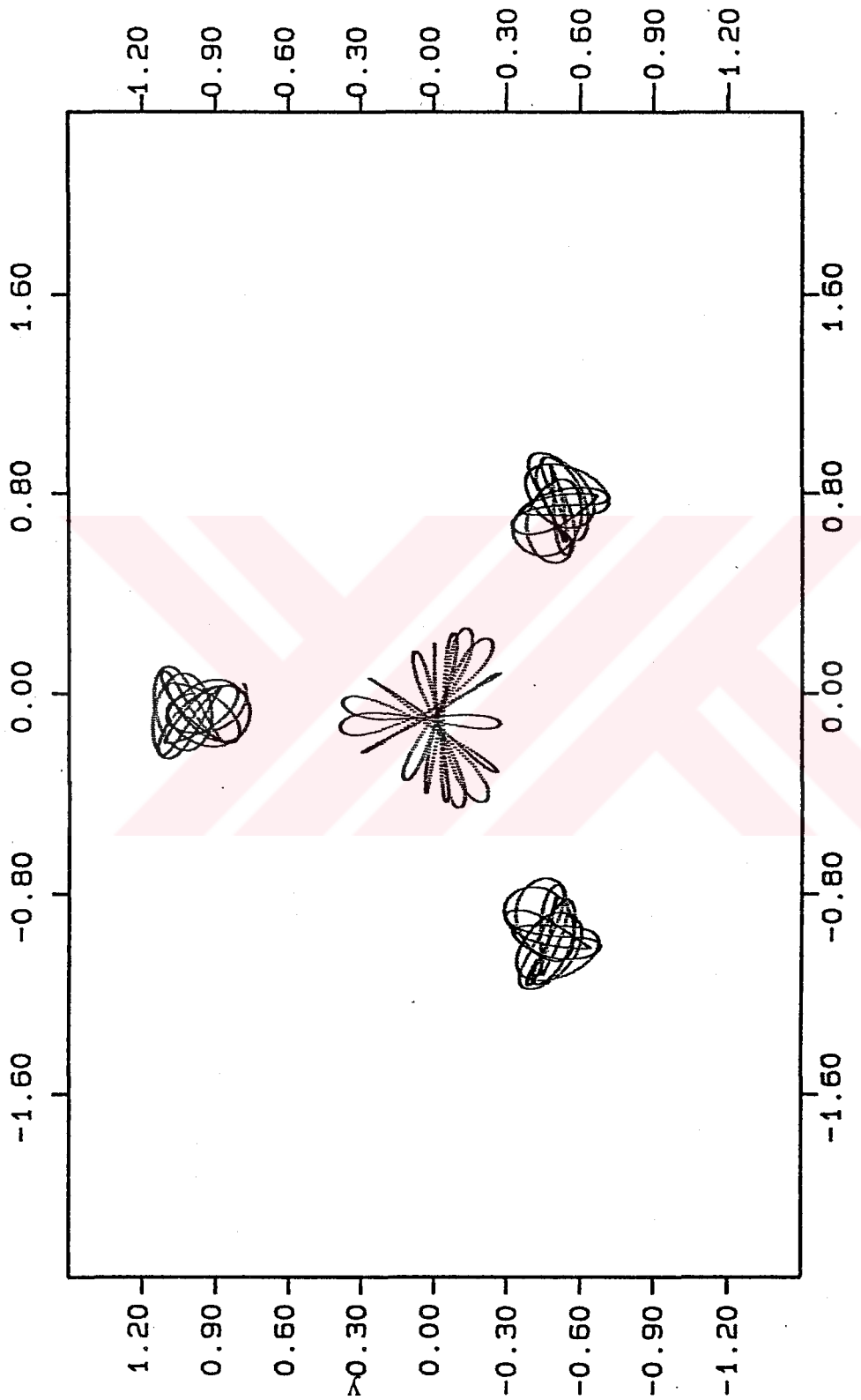


Fig. (5.4) Stability of fix point for four vortices

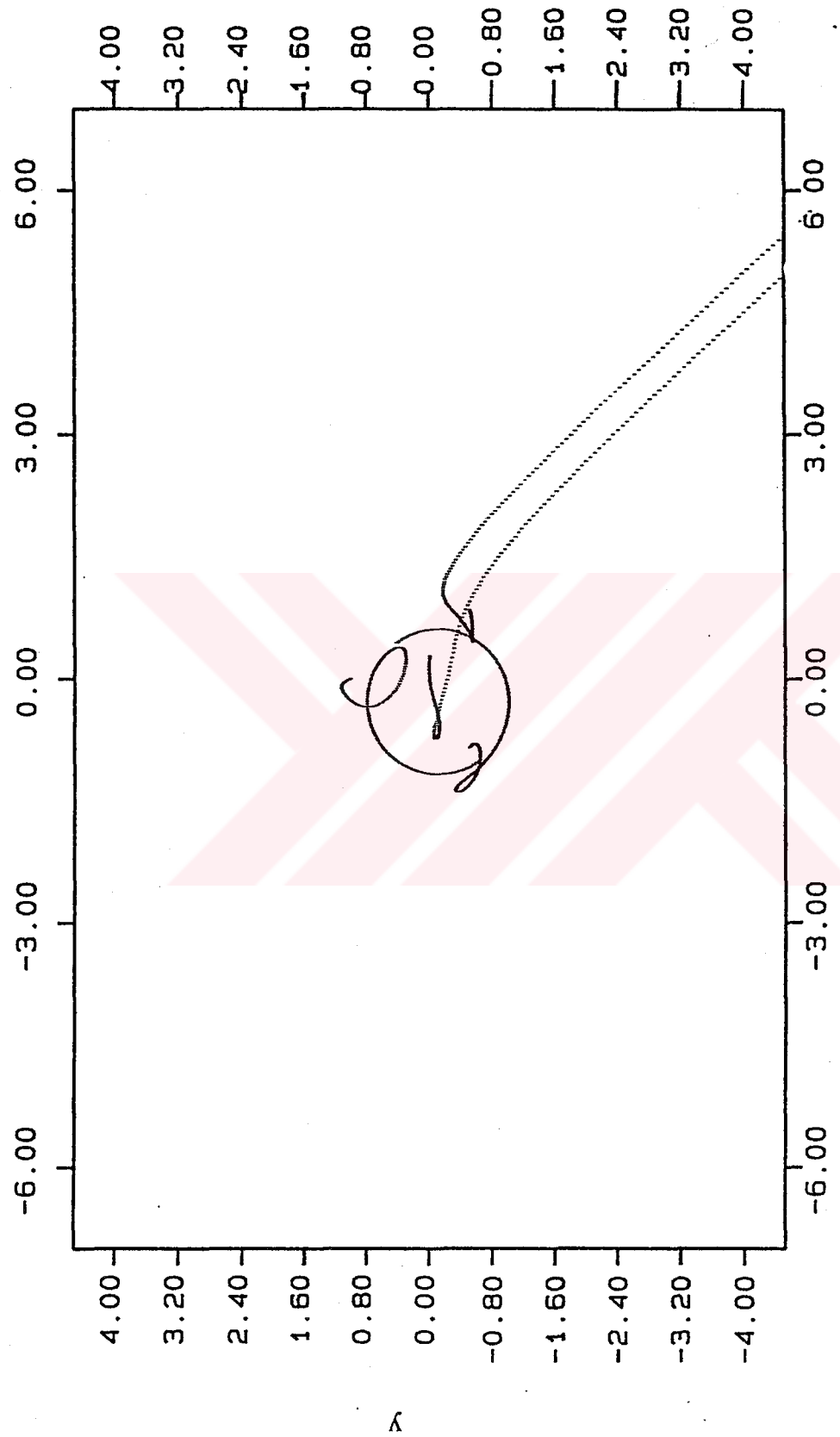


Fig. (5.5) Stability of fix point for four vortices, large perturbation

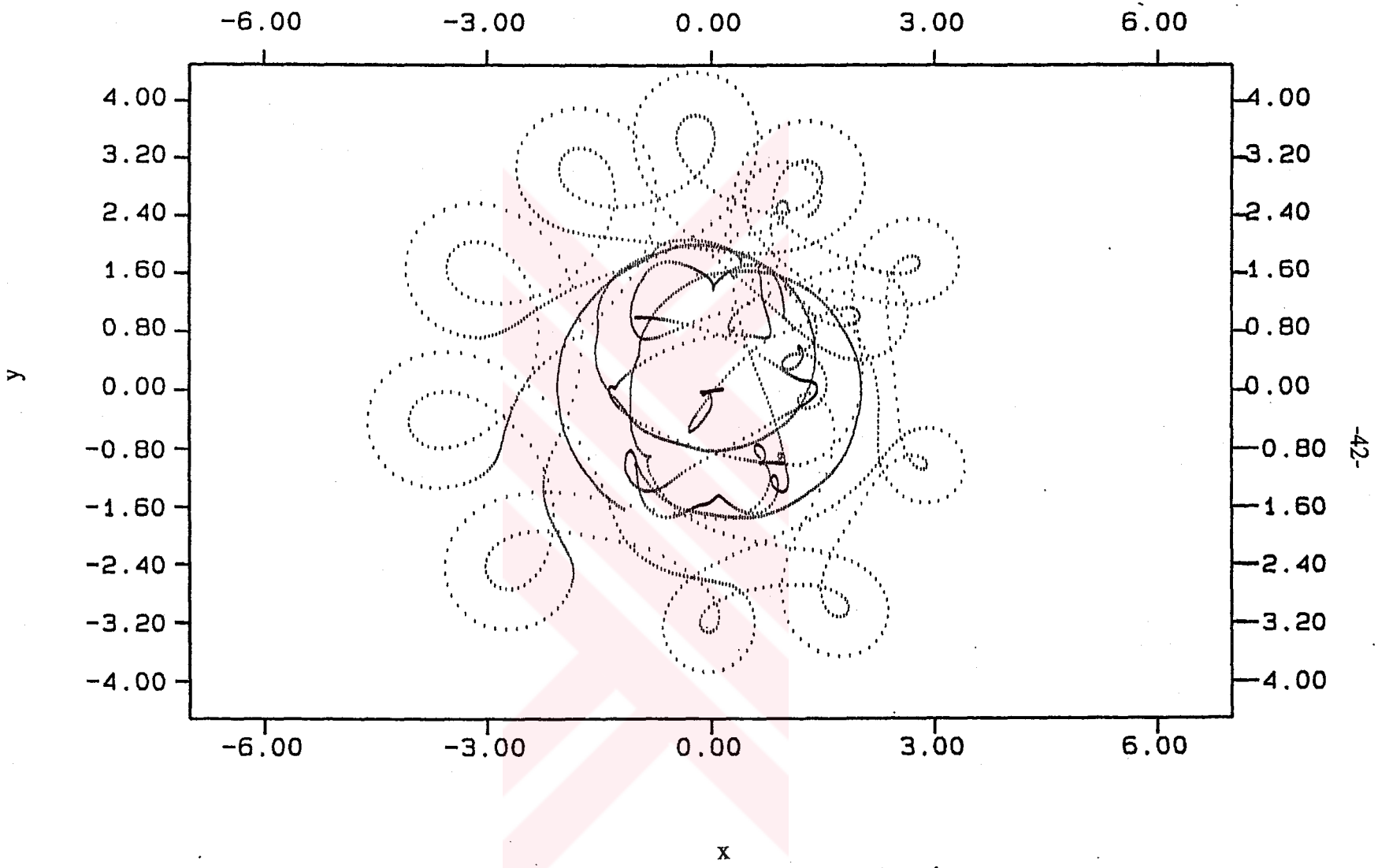


Fig. (5.6) Stability of fix point for five vortices

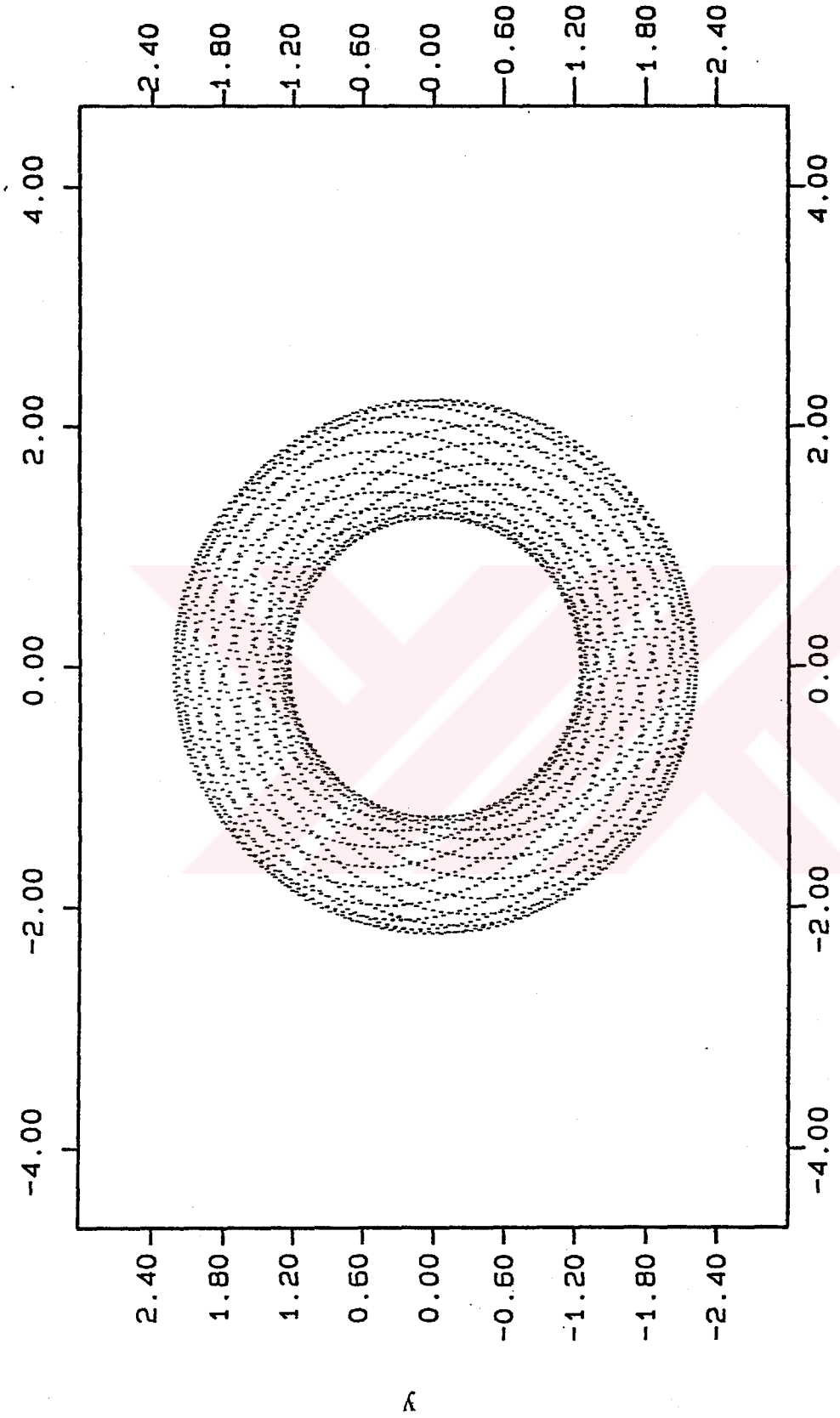


Fig. (5.7) Unbounded, rectangular configuration, trajectory

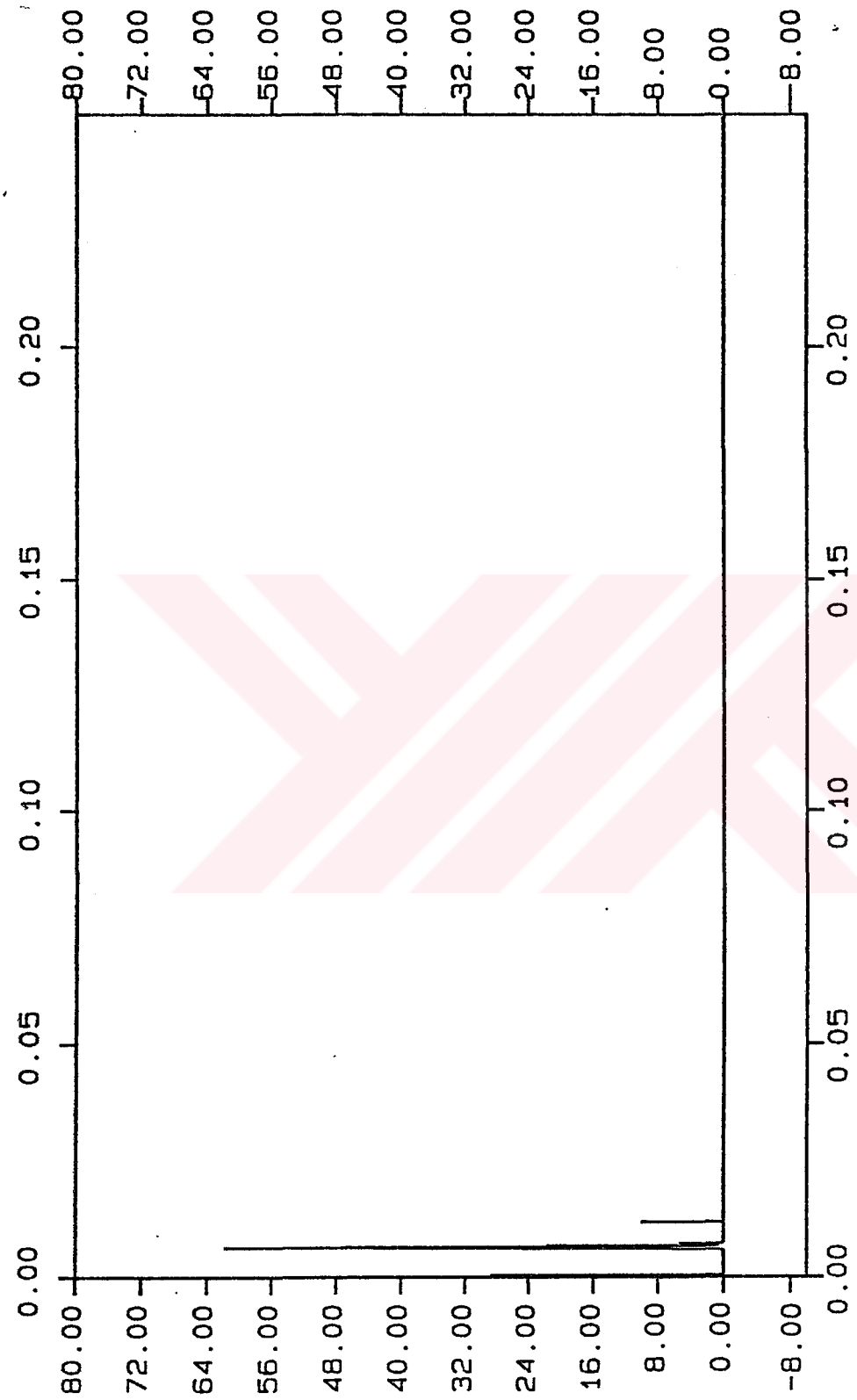


Fig. (5.8) Unbounded, rectangular configuration, power spectrum

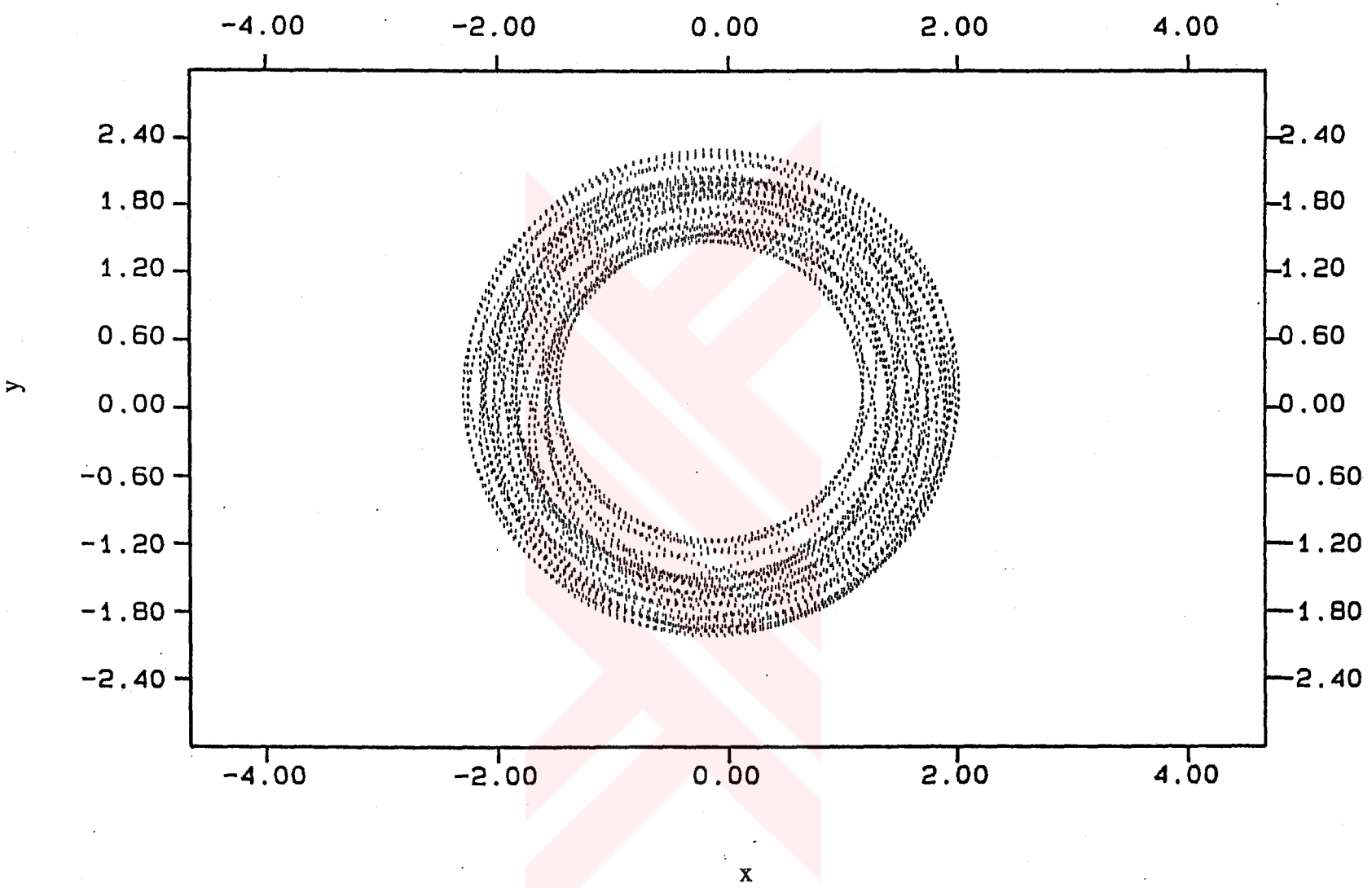


Fig. (5.9) Unbounded, slightly perturbed rectangular configuration, trajectory

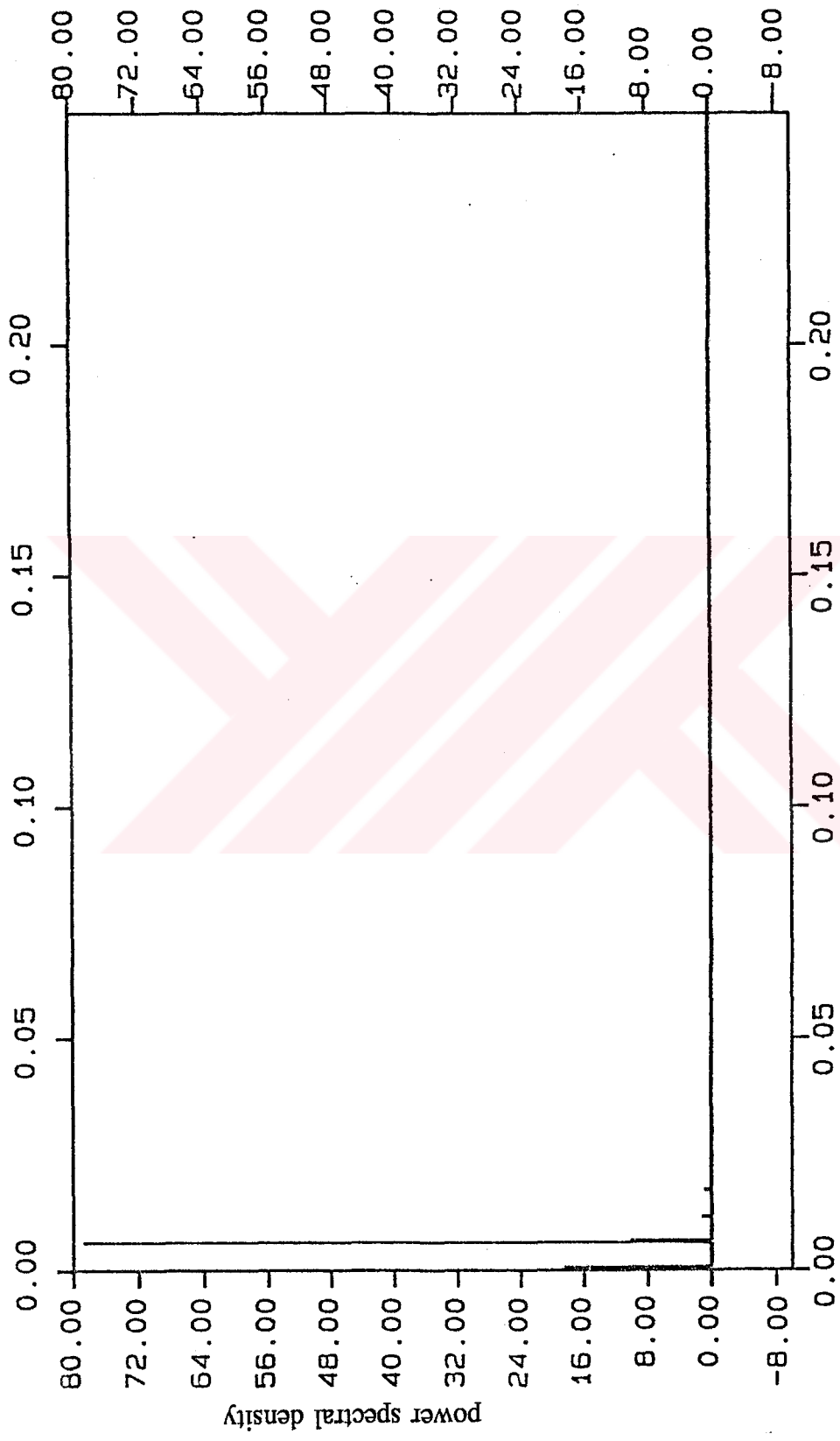


Fig. (5.10) Unbounded, slightly perturbed rectangular configuration, power spectrum

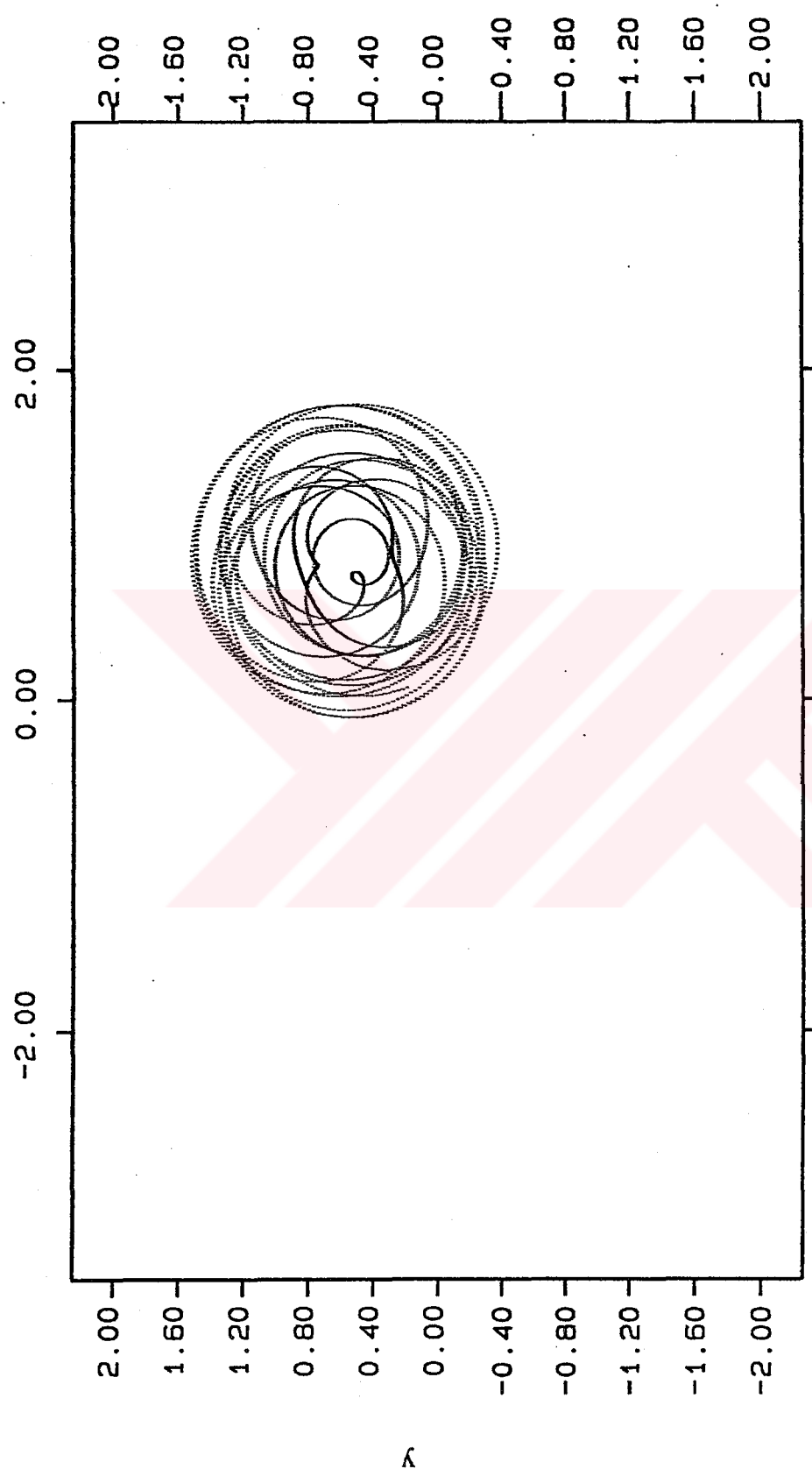


Fig. (5.11) Unbounded, perturbed rectangular configuration, trajectory

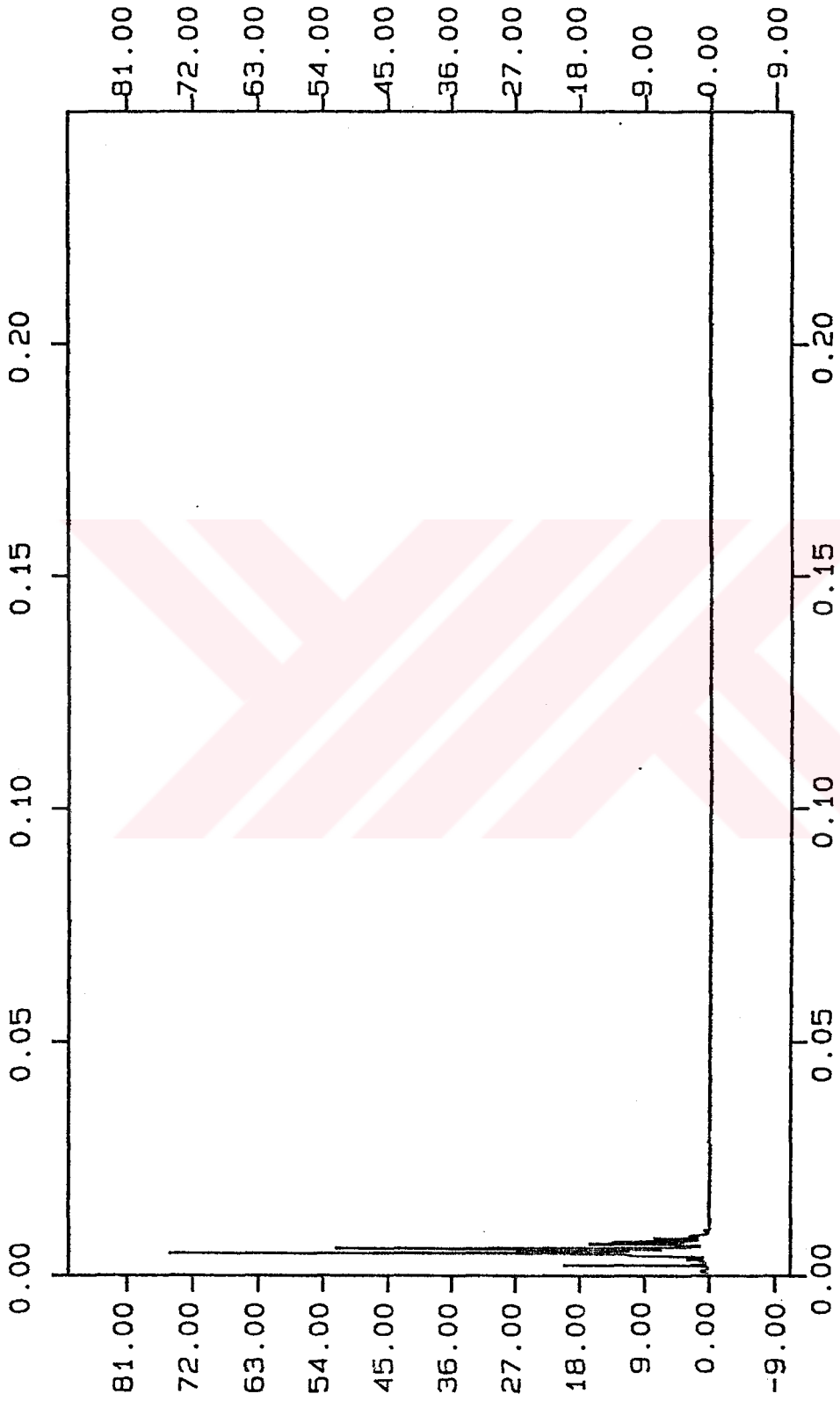


Fig. (5.12) Unbounded, perturbed rectangular configuration, power spectrum

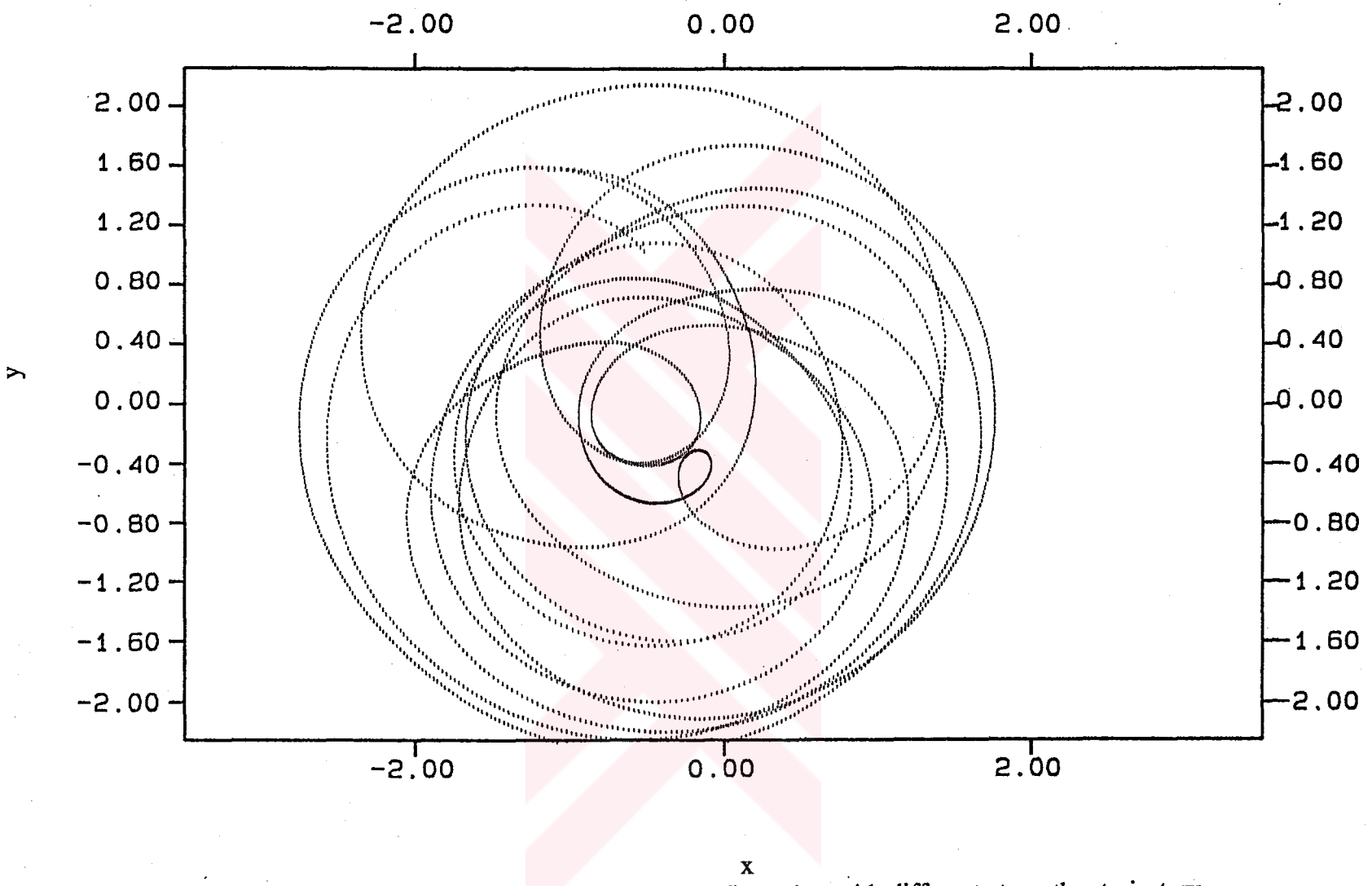


Fig. (5.13) Unbounded, rectangular configuration with different strengths, trajectory

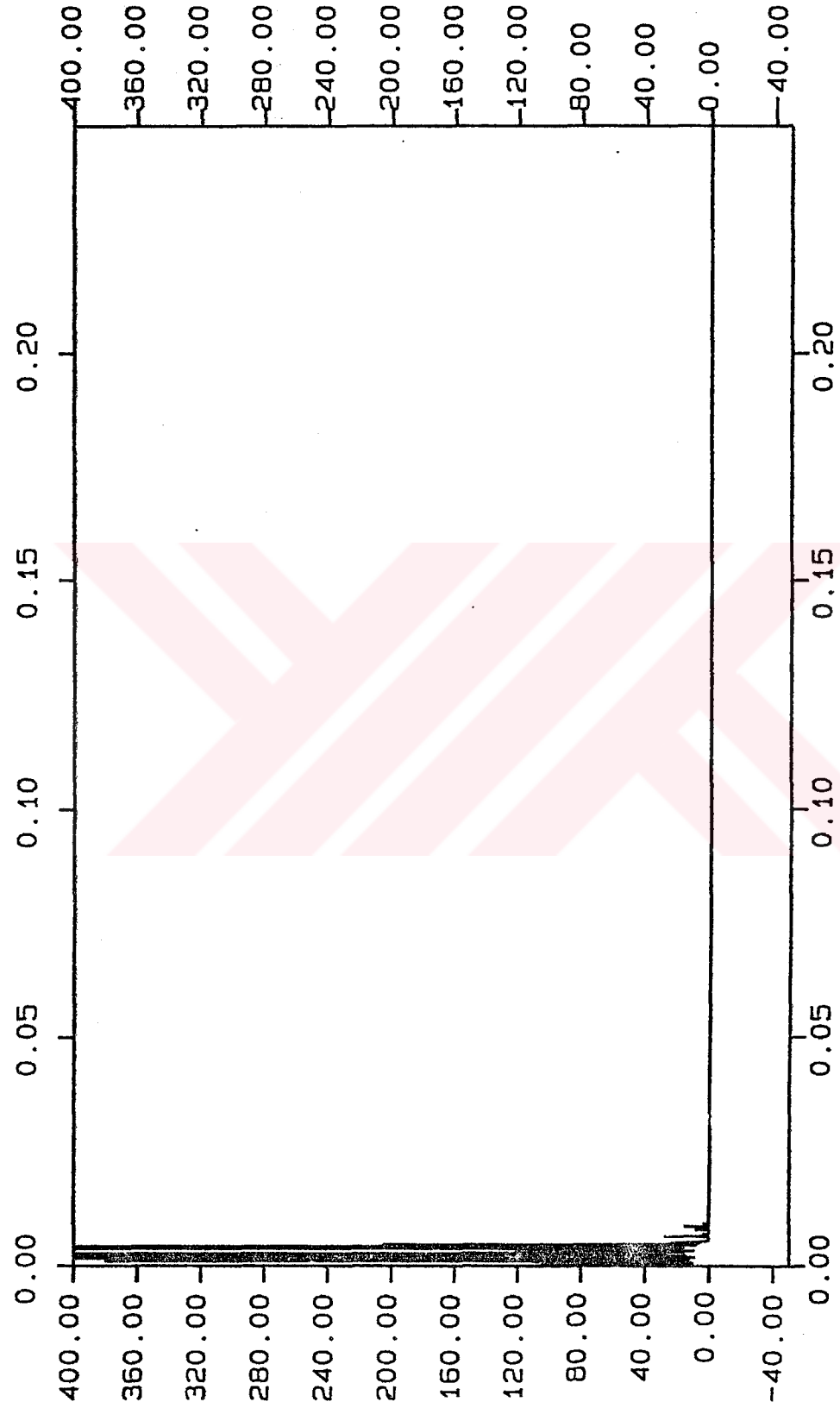


Fig. (5.14) Unbounded, rectangular configuration with different strengths, power spectrum

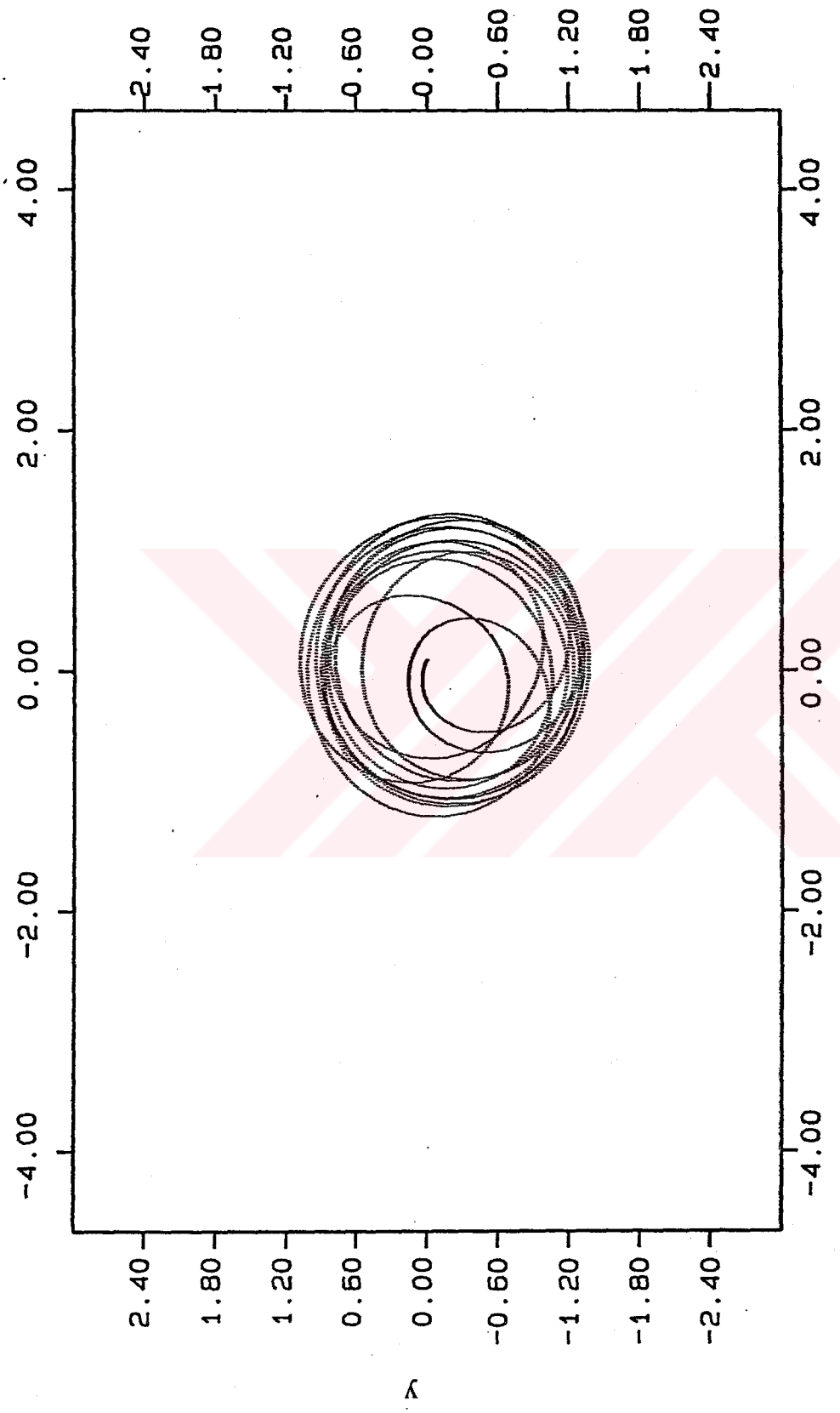


Fig. (5.15) Unbounded, perturbed triangular configuration, trajectory

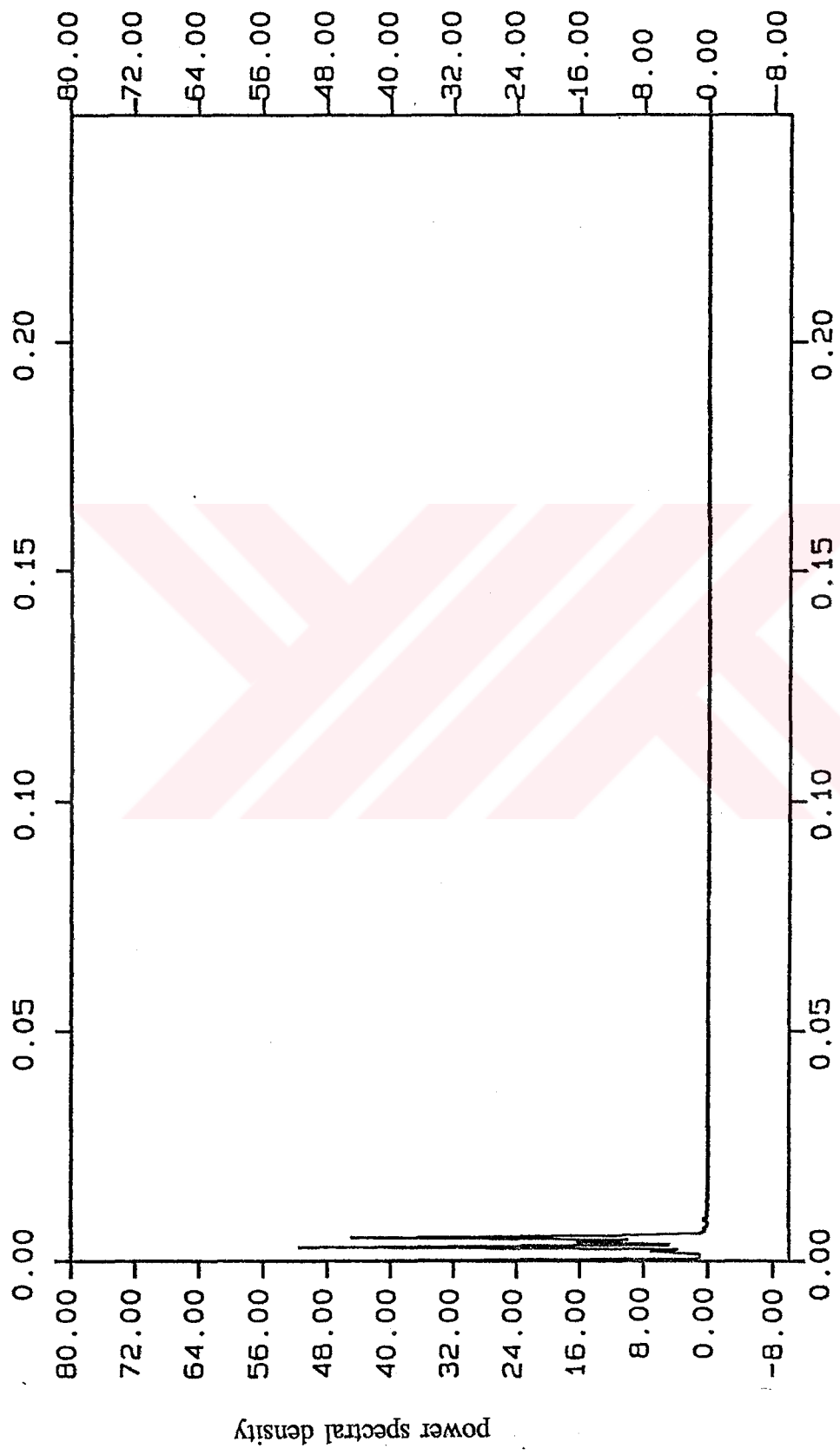


Fig. (5.16) Unbounded, perturbed triangular configuration, power spectrum

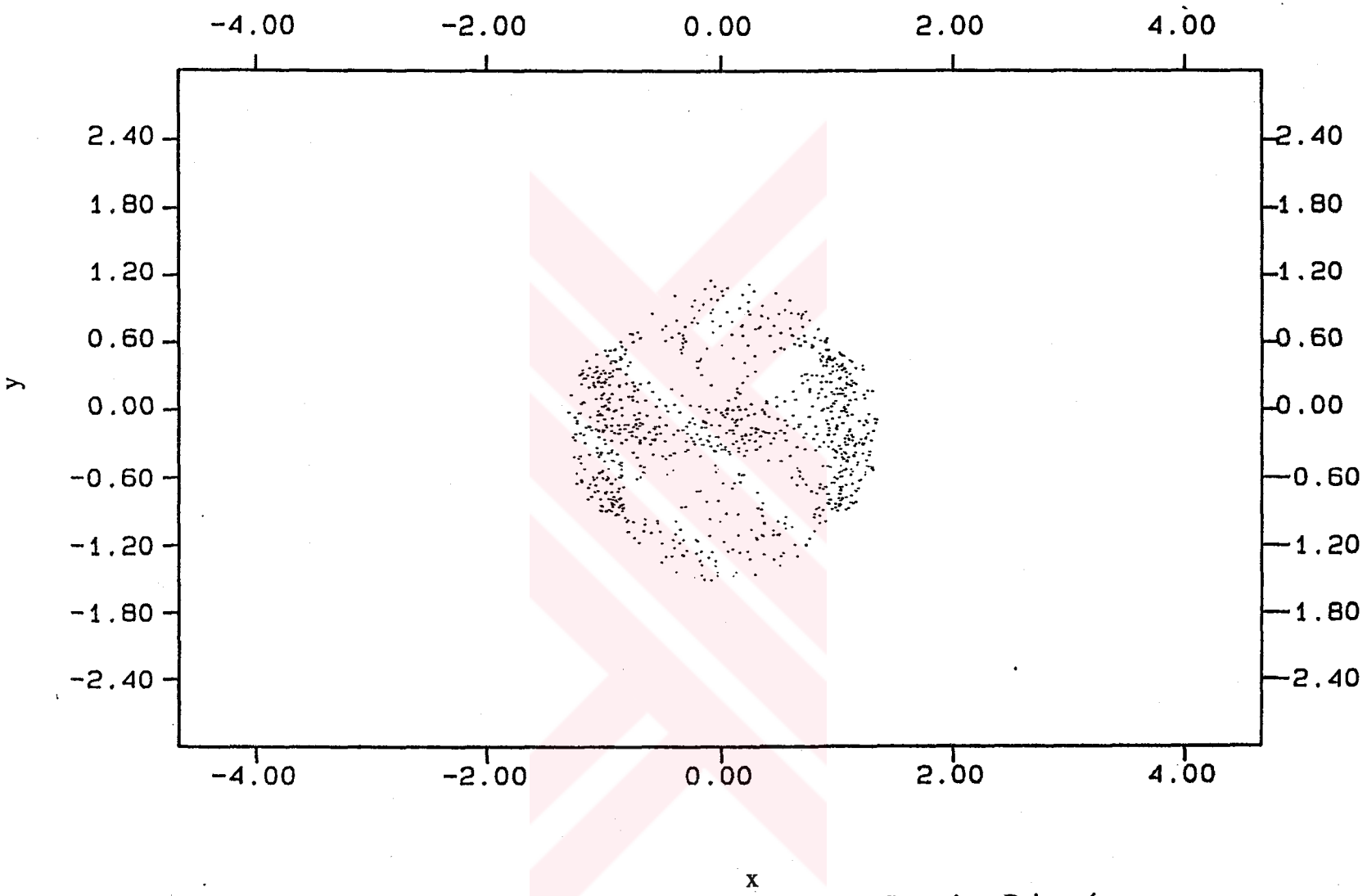


Fig. (5.17) Unbounded, perturbed triangular configuration, Poincaré map

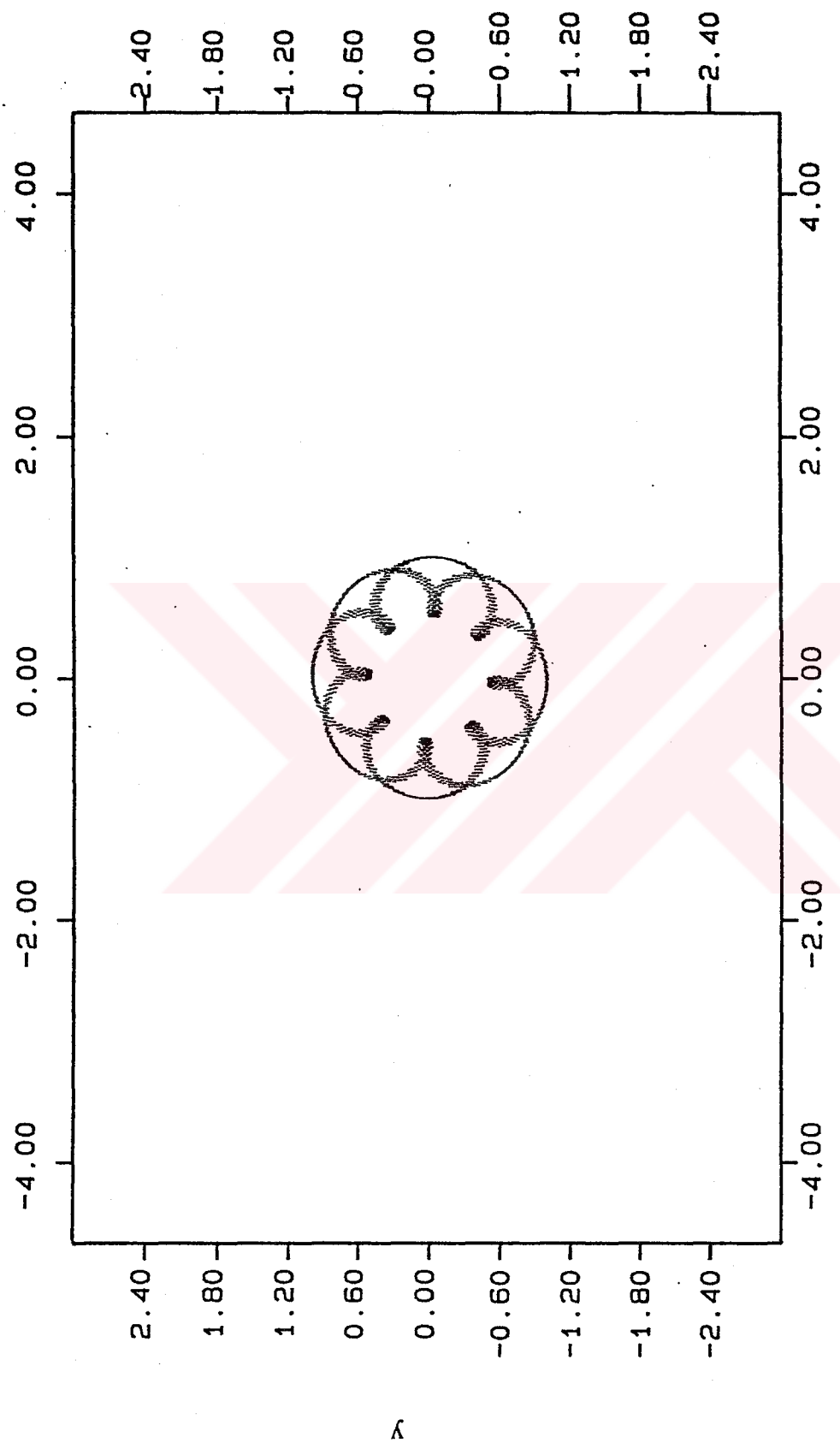


Fig. (5.18) Unbounded, symmetric line configuration, trajectory

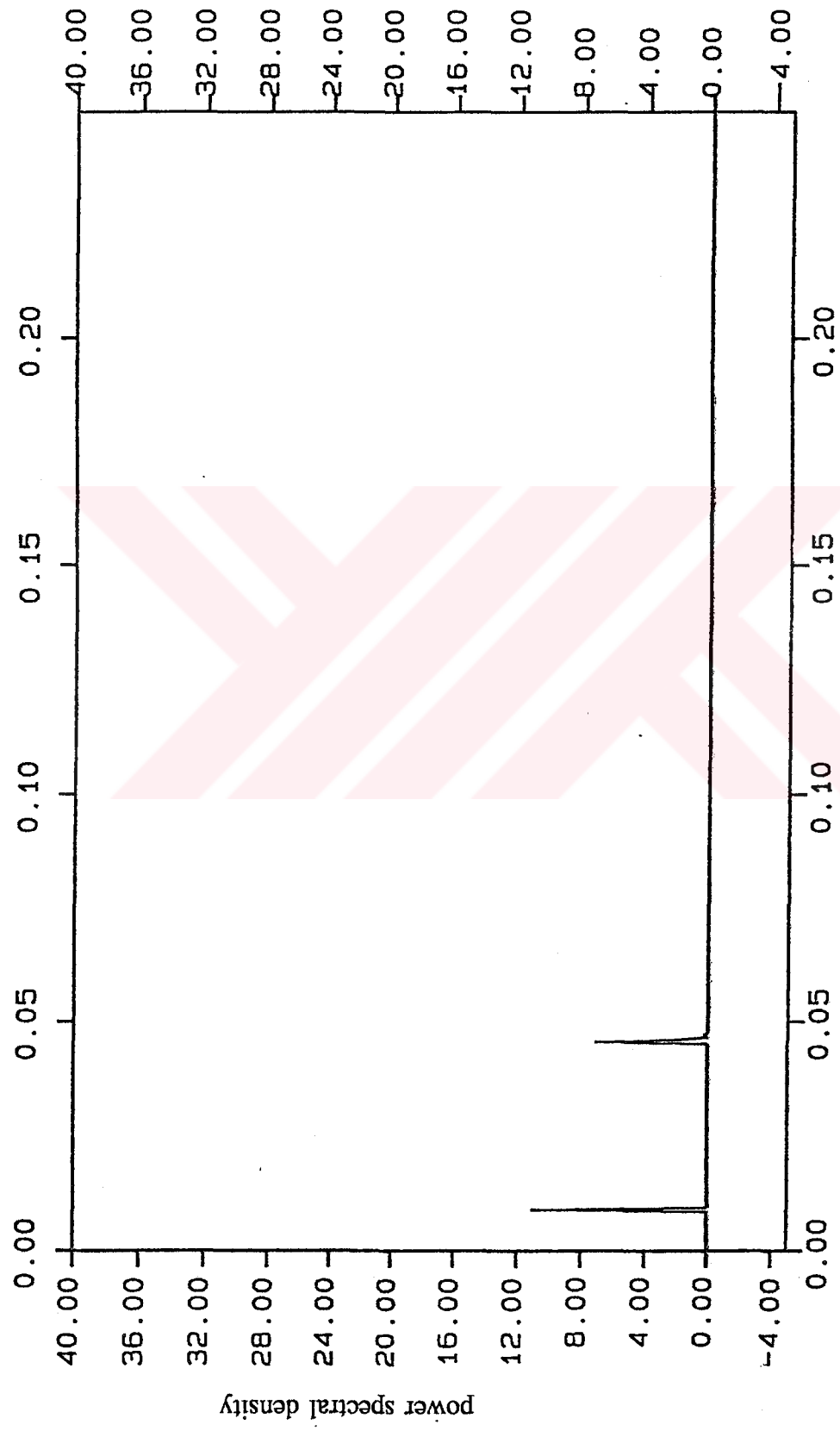


Fig. (5.19) Unbounded, symmetric line configuration, power spectrum

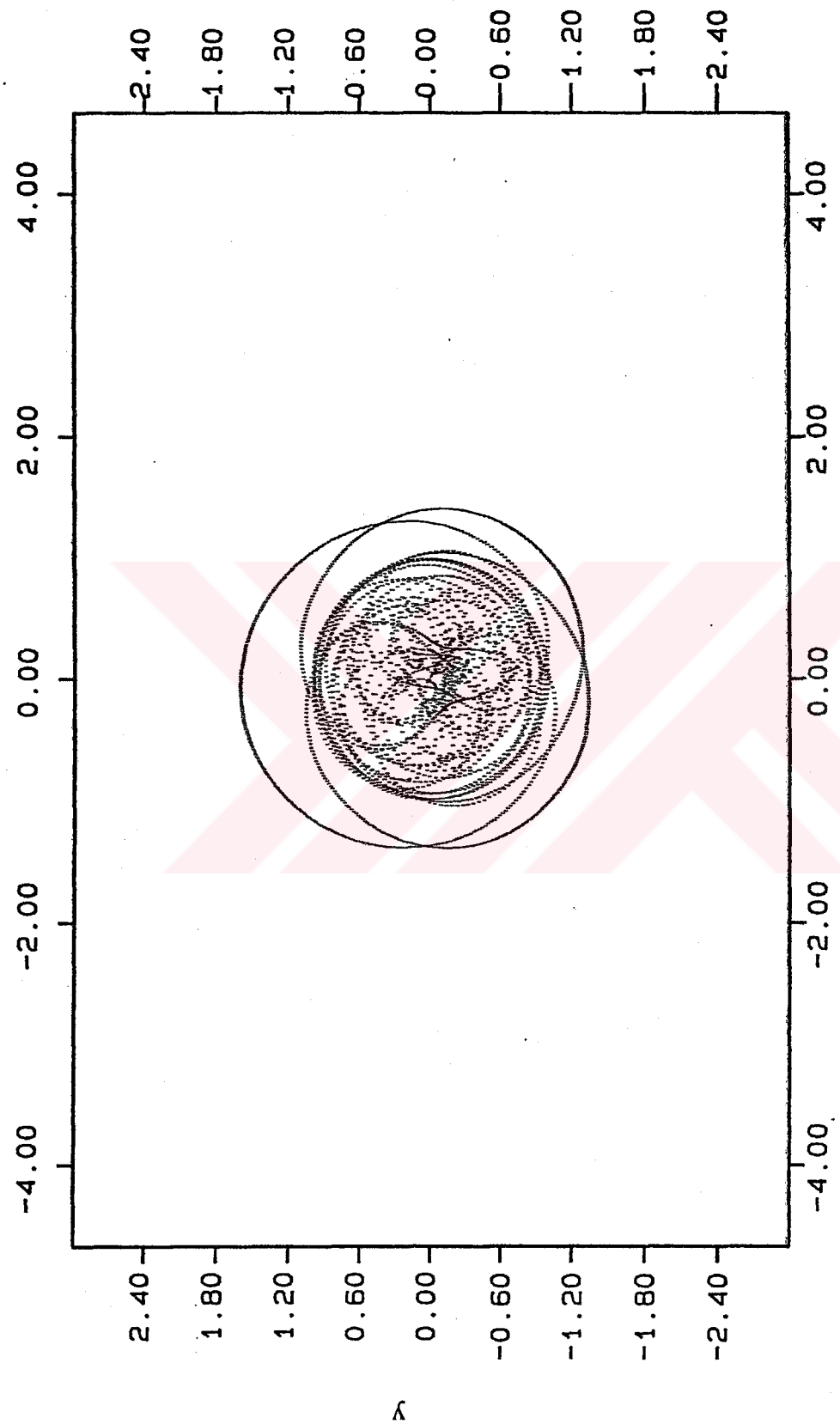


Fig. (5.20) Unbounded, chaotic, line configuration, trajectory

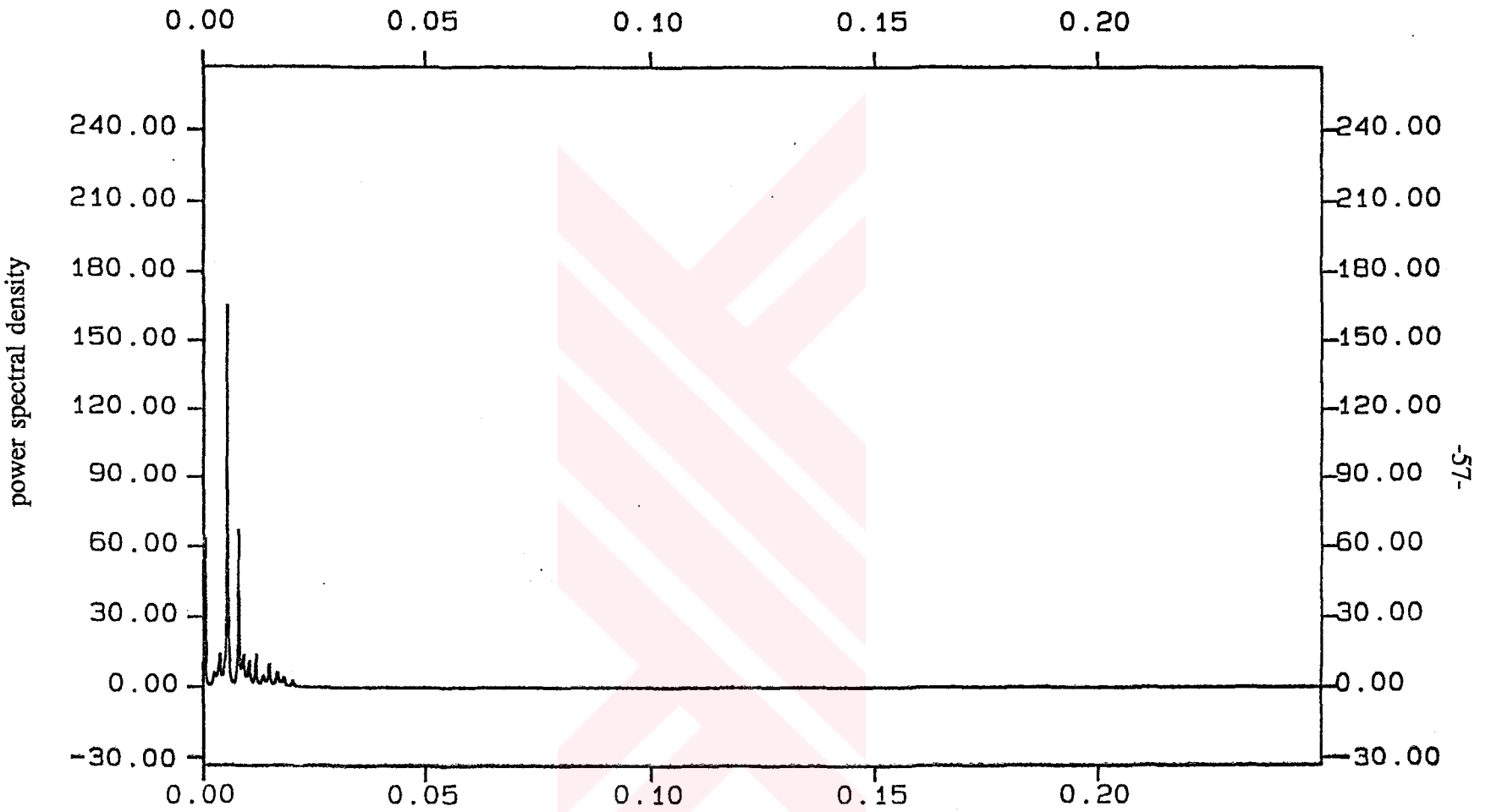


Fig. (5.21) Unbounded, chaotic, line configuration, power spectrum

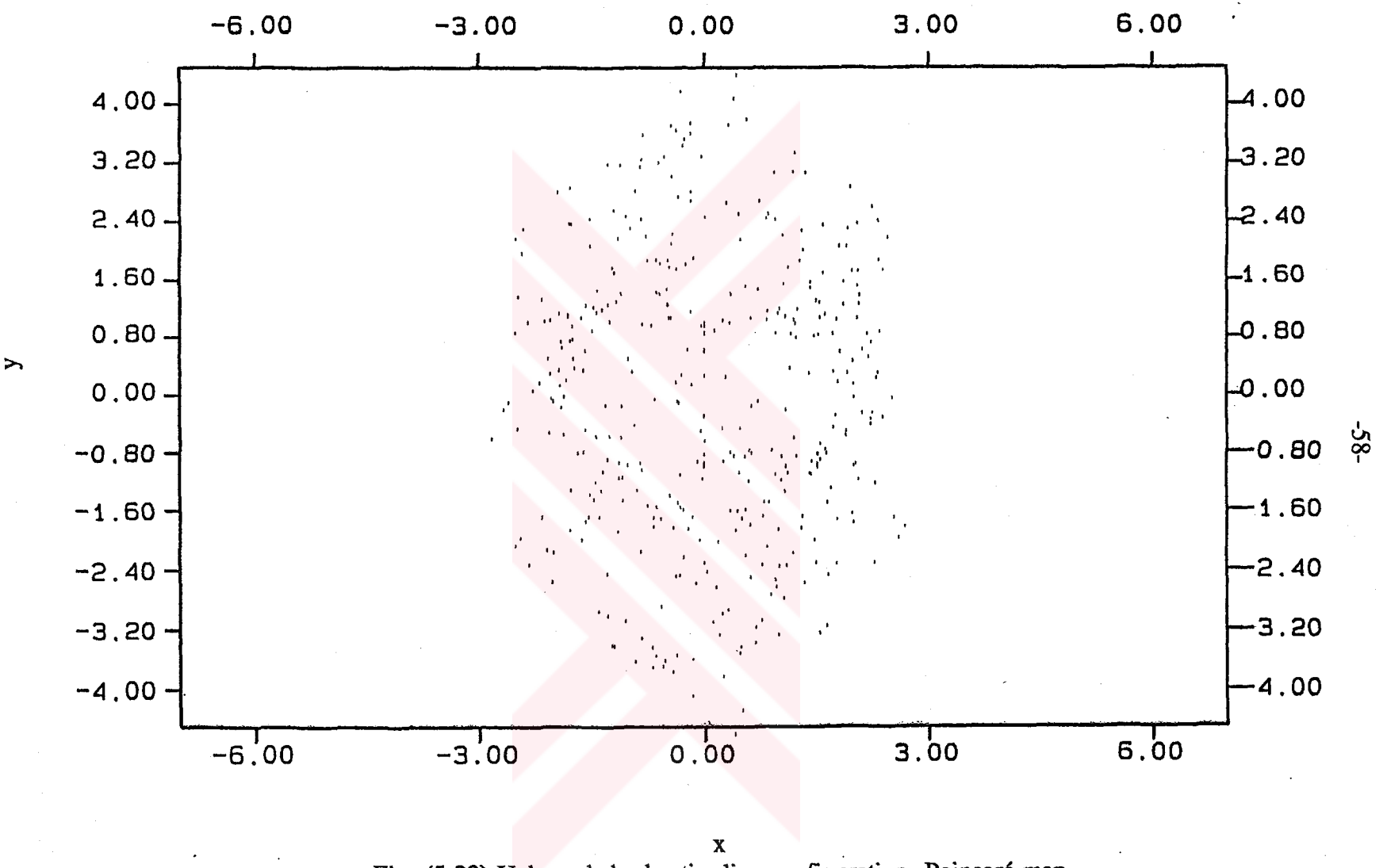


Fig. (5.22) Unbounded, chaotic, line configuration, Poincaré map

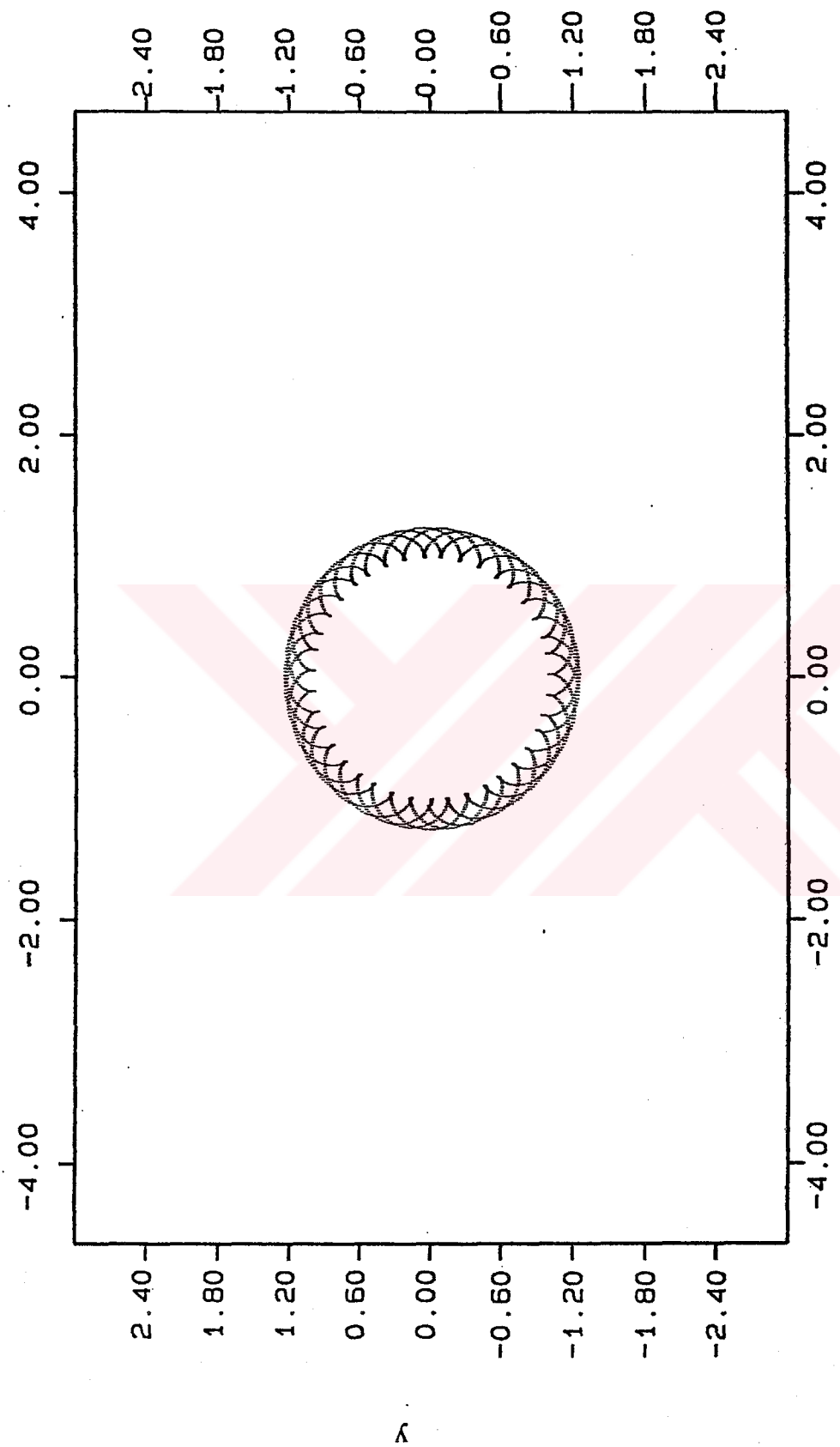


Fig. (5.23) Unbounded, integrable, line configuration, trajectory

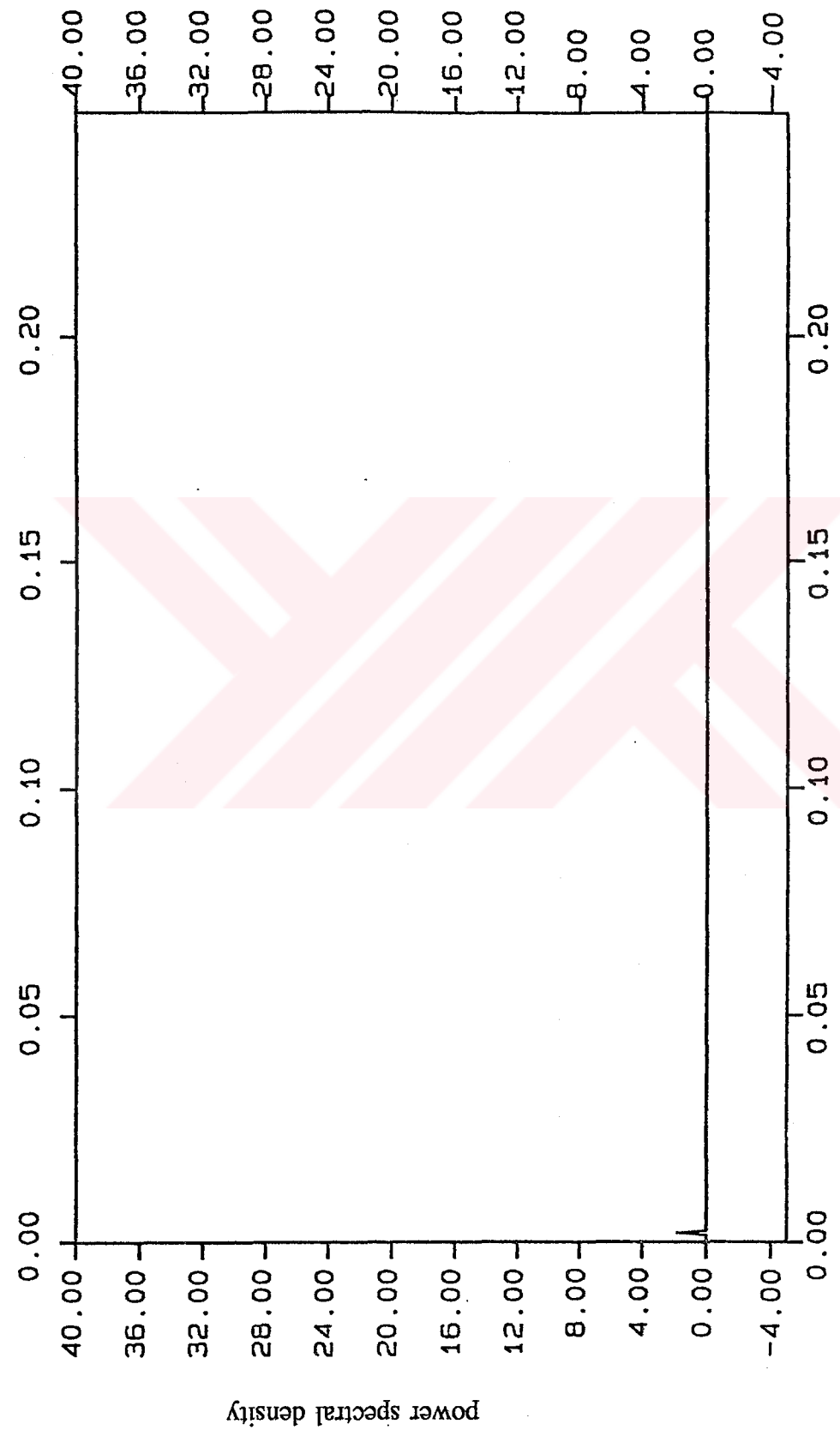


Fig. (5.24) Unbounded, integrable, line configuration, power spectrum

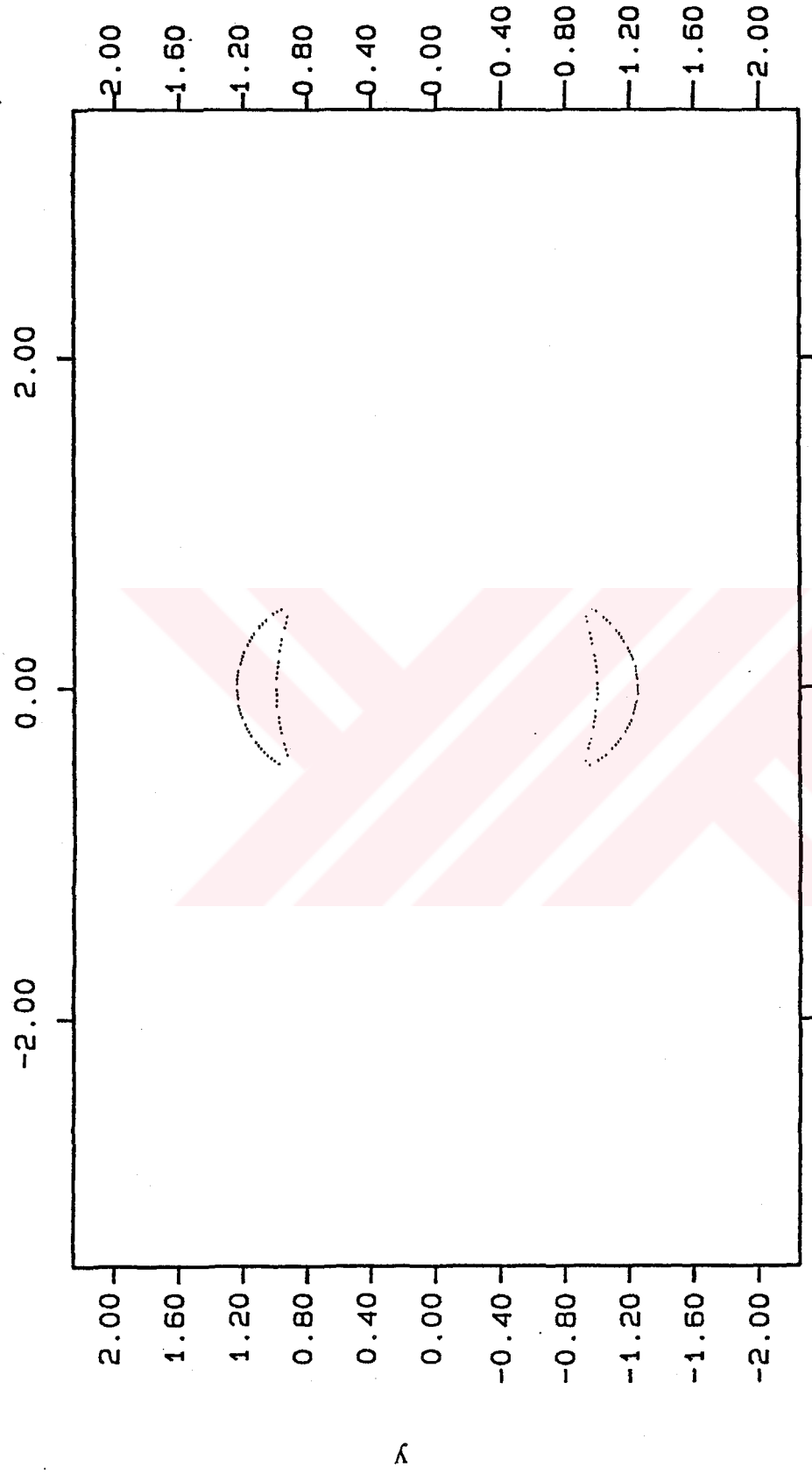


Fig. (5.25) Unbounded, integrable, line configuration, Poincaré map

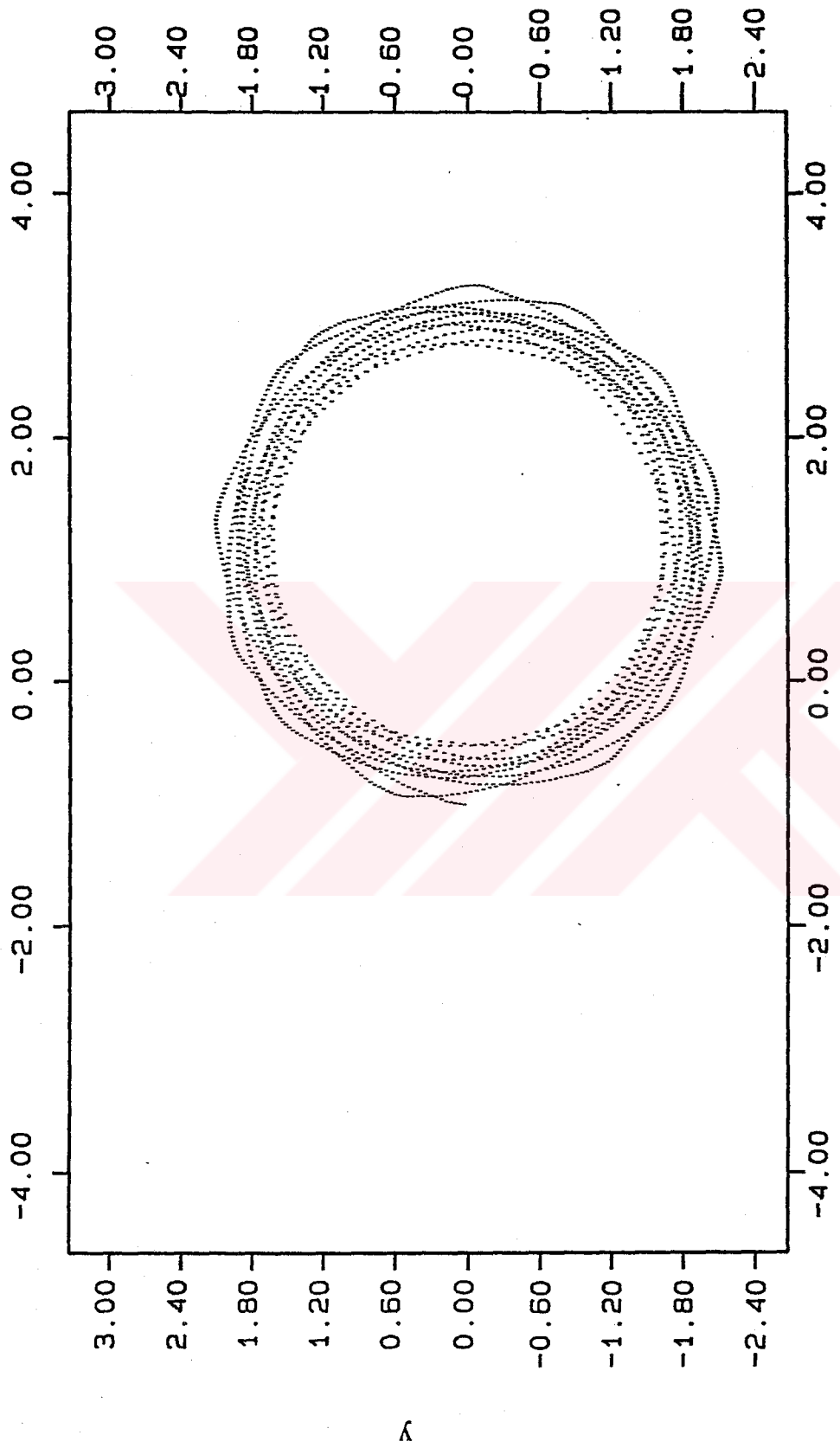


Fig. (5.26) Unbounded, integrable, line configuration, trajectory

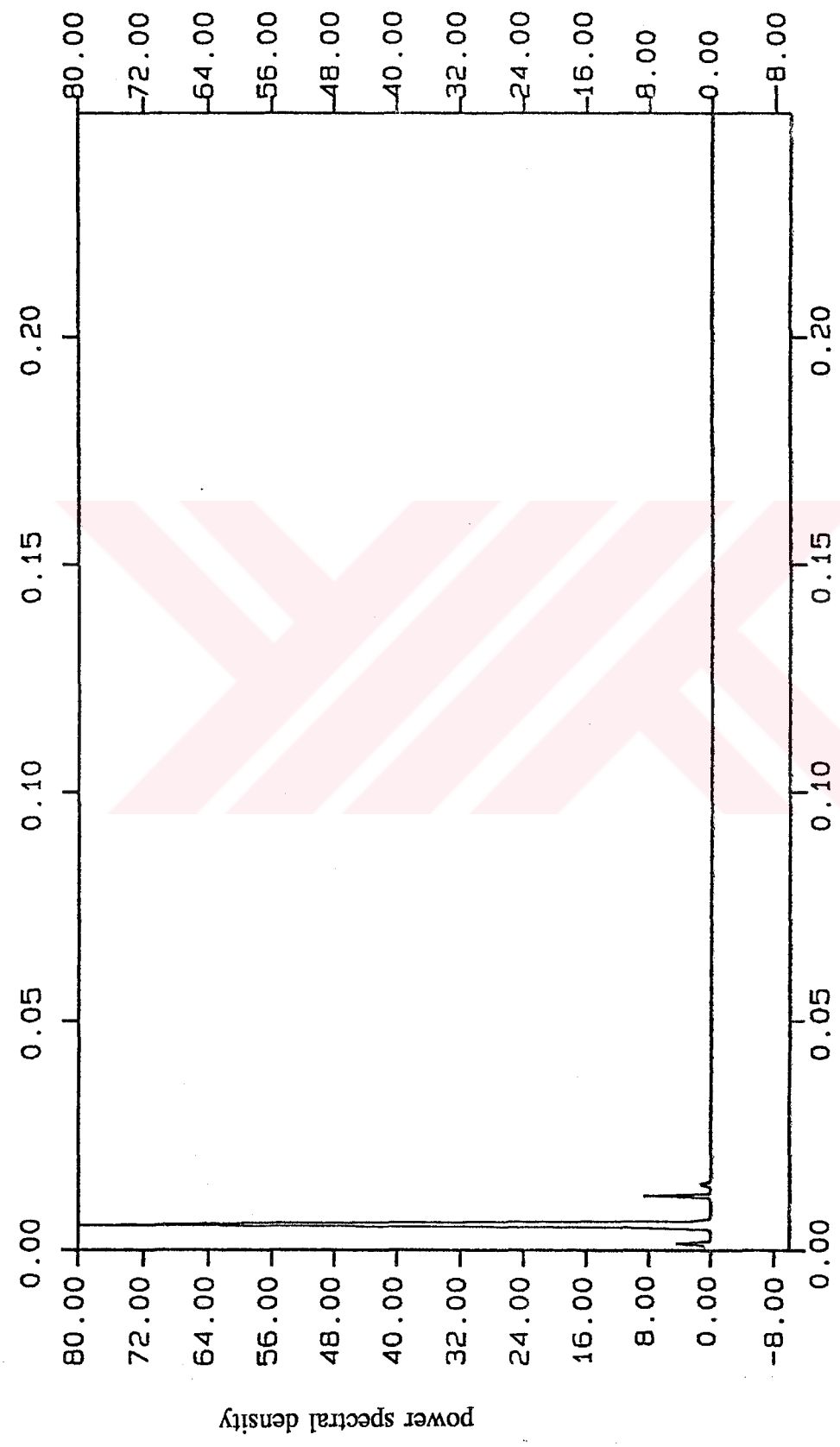


Fig. (5.27) Unbounded, integrable, line configuration, power spectrum

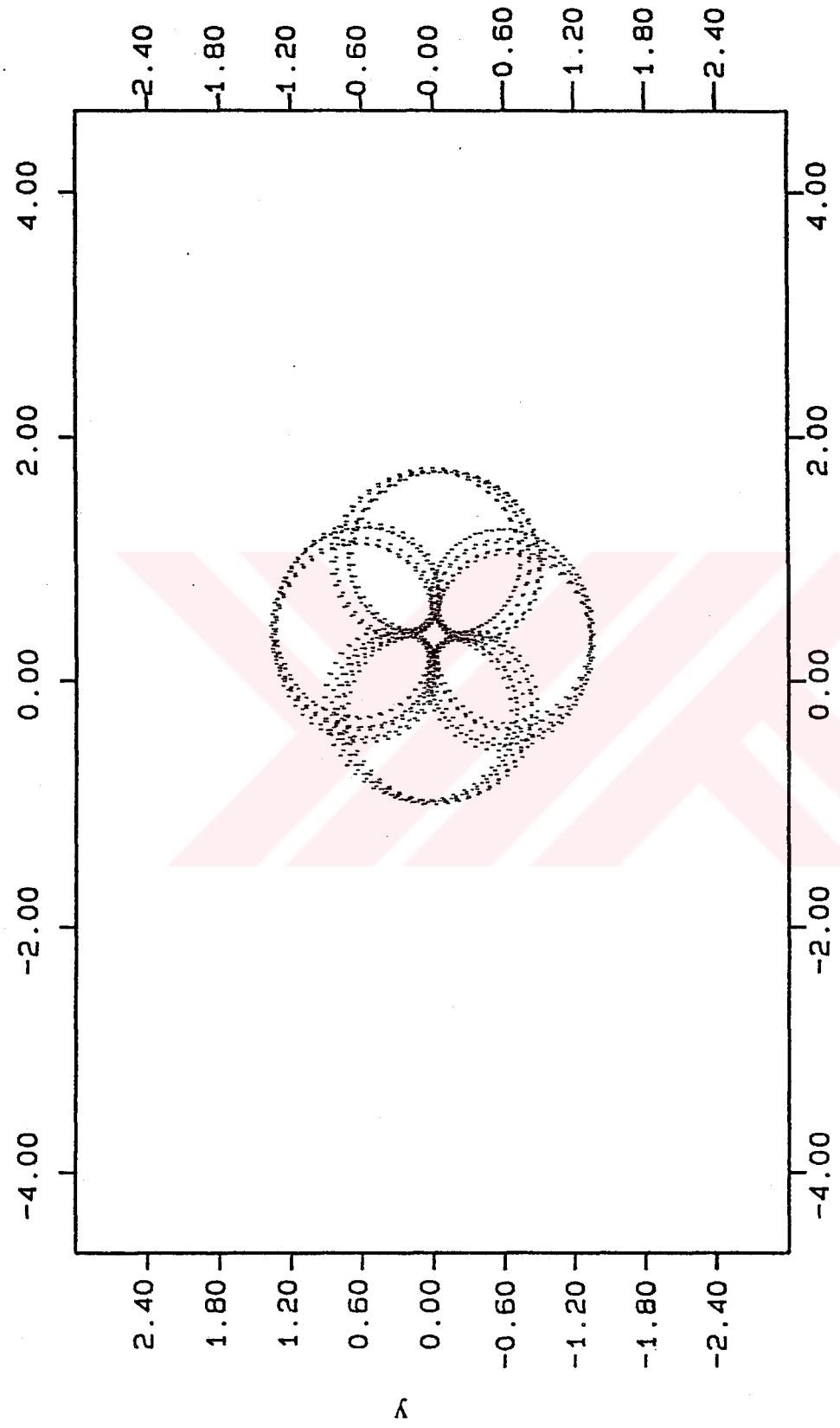


Fig. (5.28) Unbounded, integrable, line configuration, trajectory

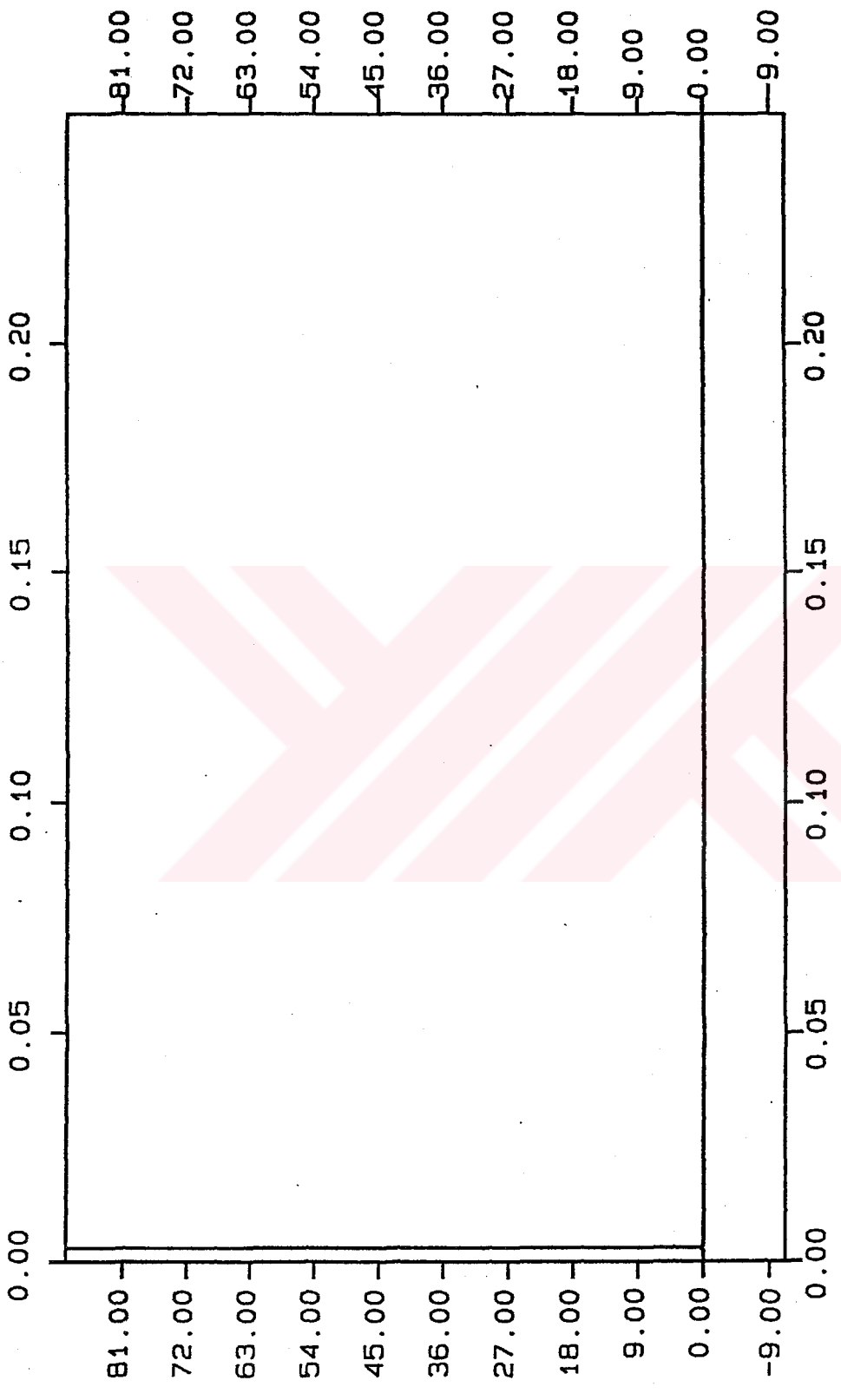


Fig. (5.29) Unbounded, integrable, line configuration, power spectrum

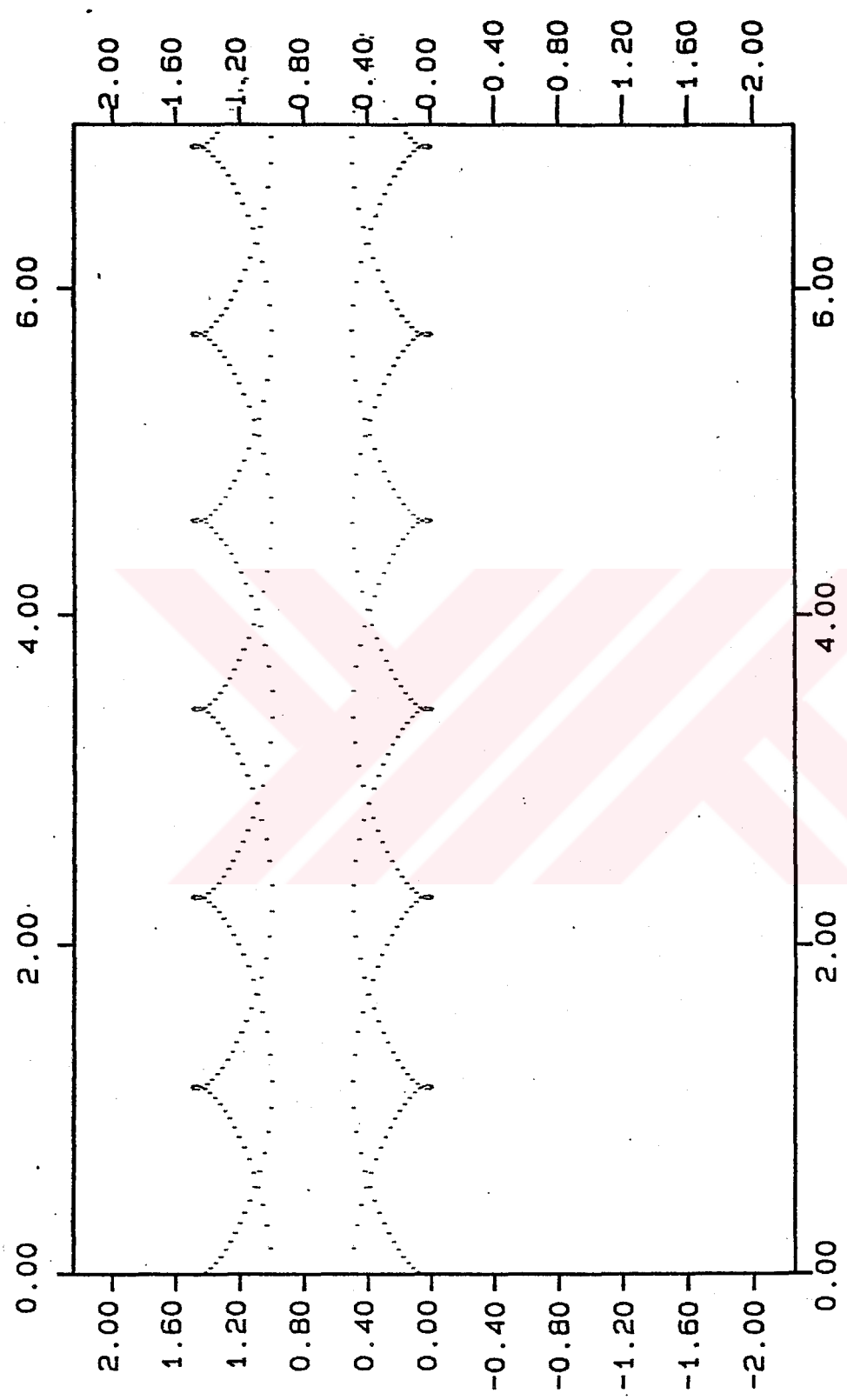


Fig. (5.30) Unbounded, integrable, line configuration of different signs, trajectory

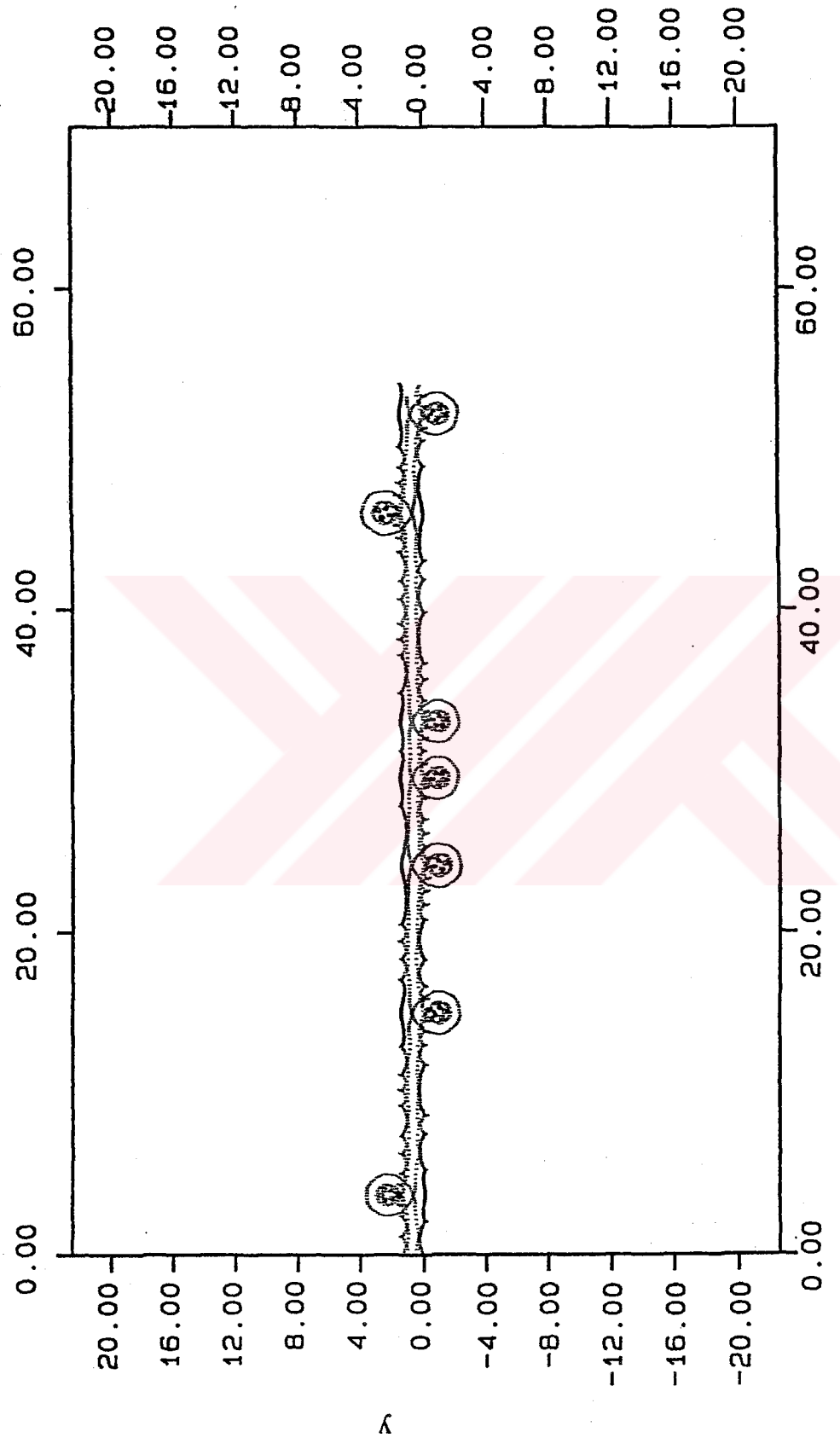


Fig. (5.31) Unbounded, chaotic, line configuration of different signs, trajectories

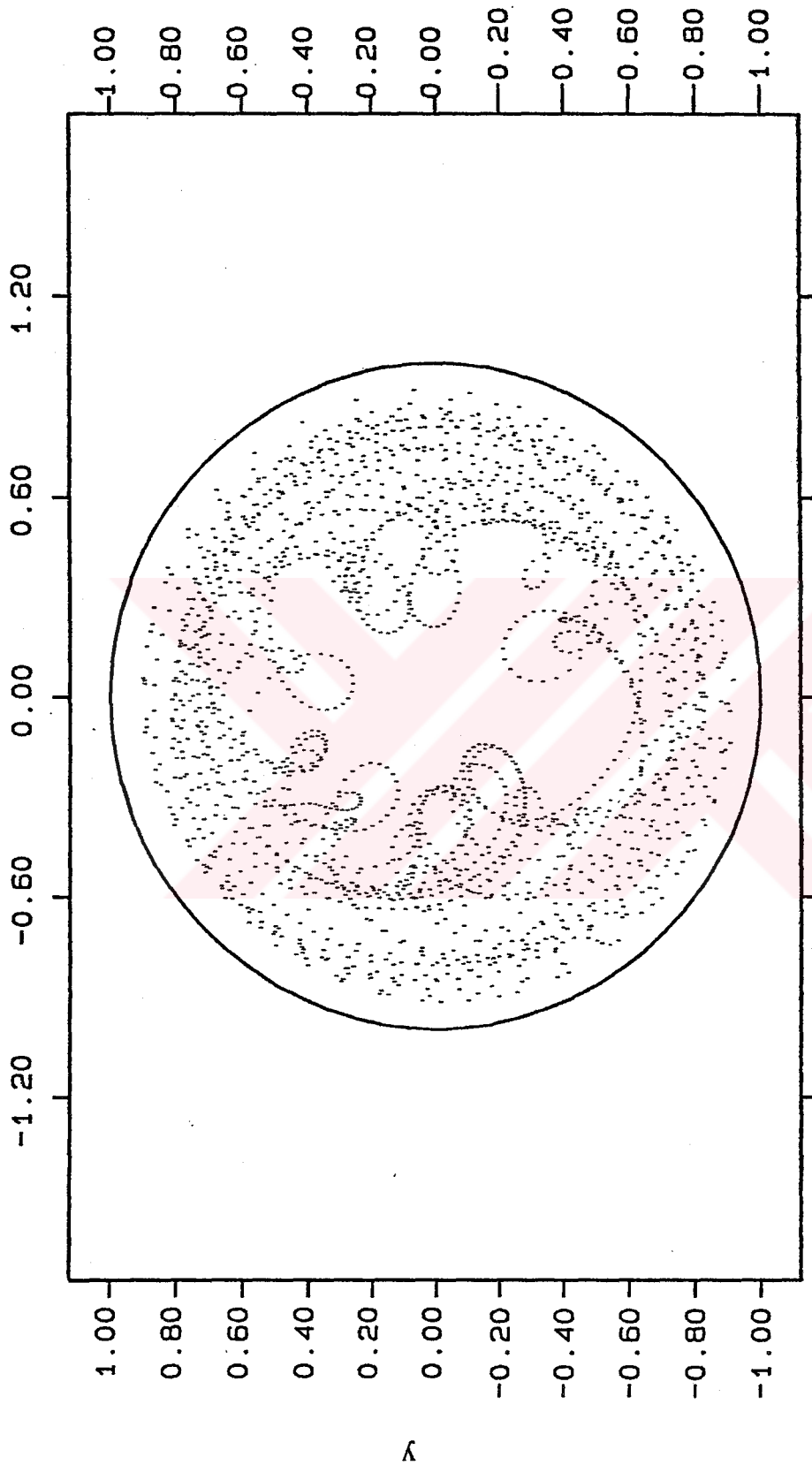


Fig. (5.32) Circular, three vortices on a line with positive signs, trajectories

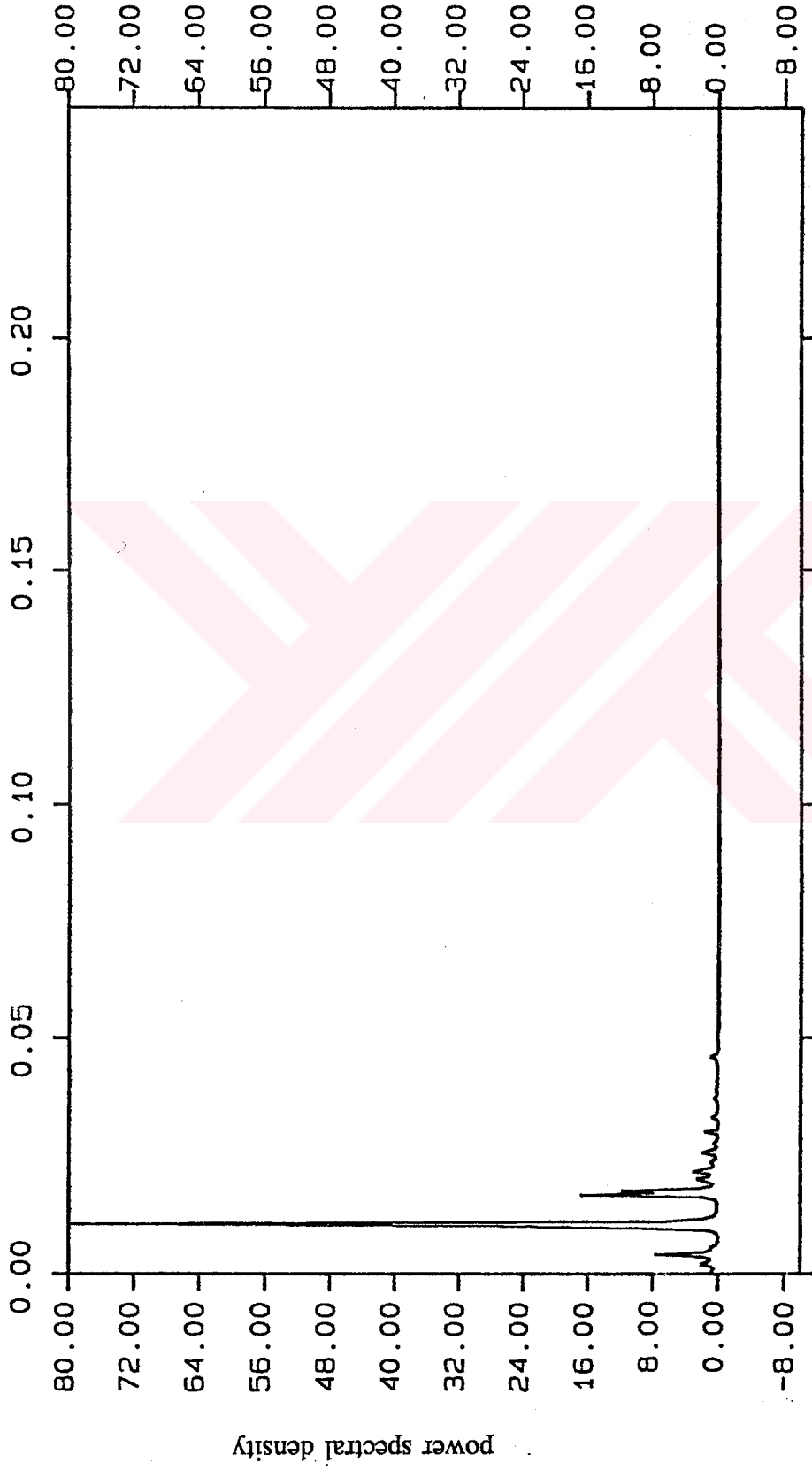


Fig. (5.33) Circular, three vortices on a line with positive signs, power spectrum

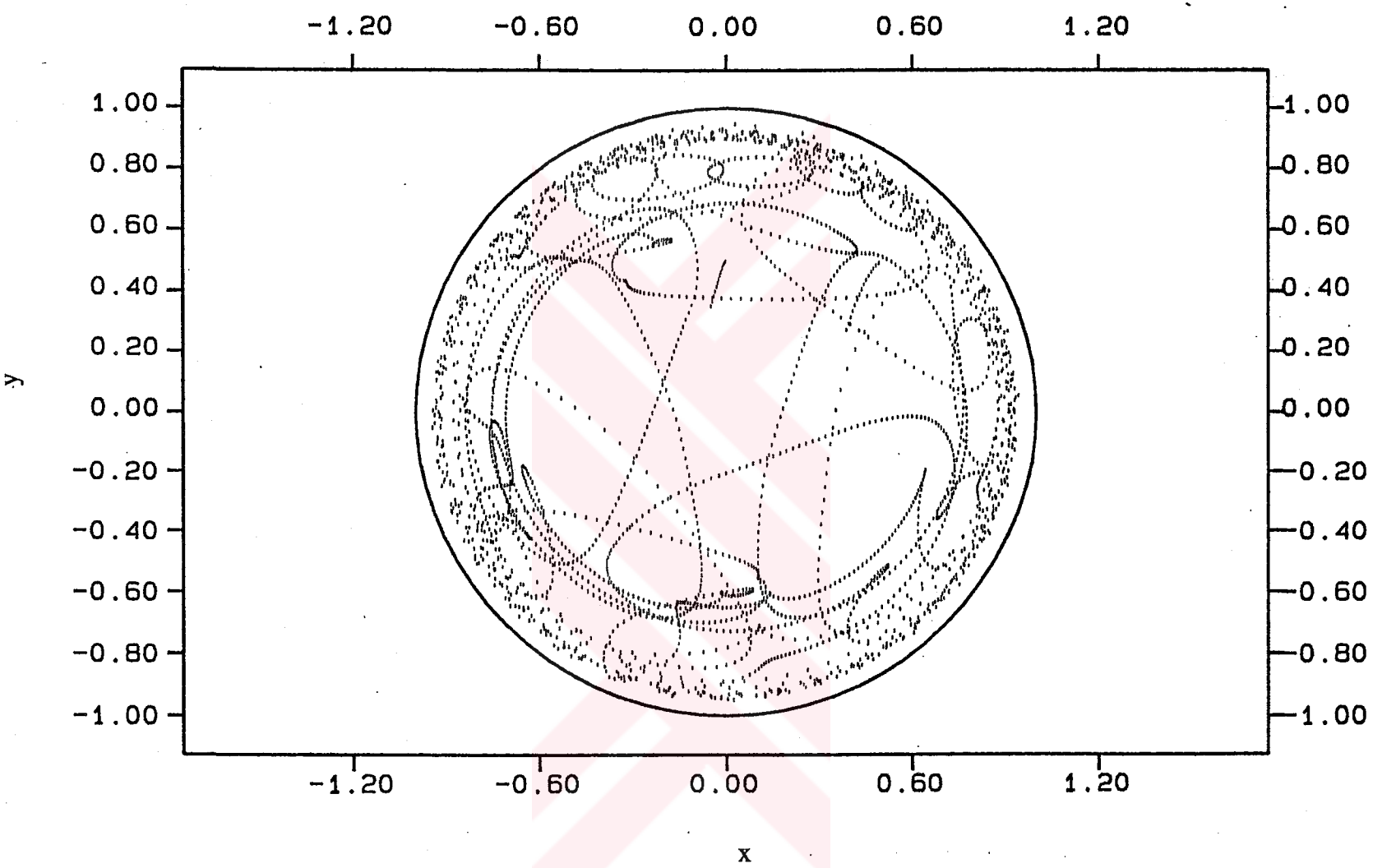


Fig. (5.34) Circular, three vortices on a line with different signs, trajectory

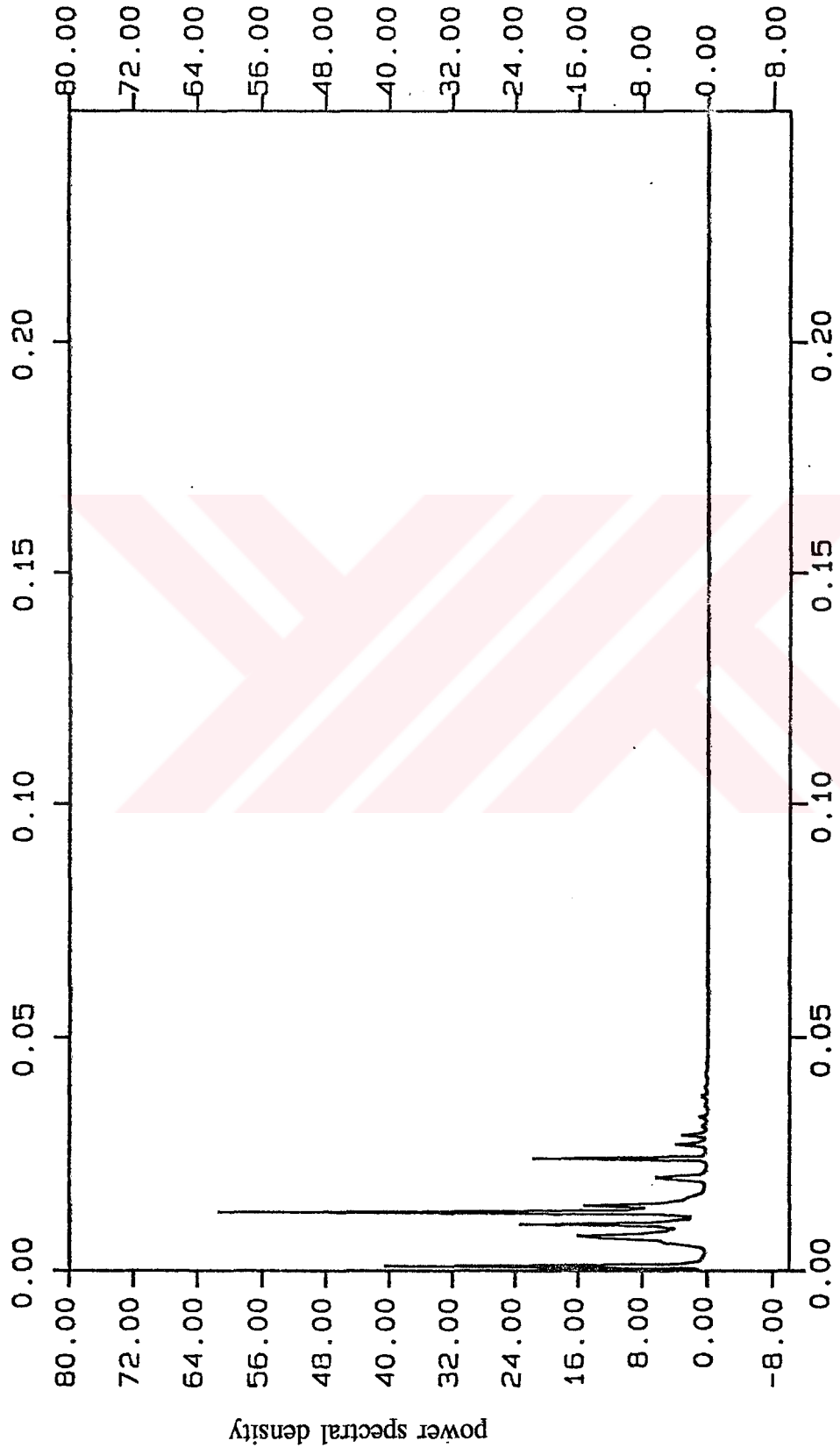


Fig. (5.35) Circular, three vortices on a line with different signs, power spectrum

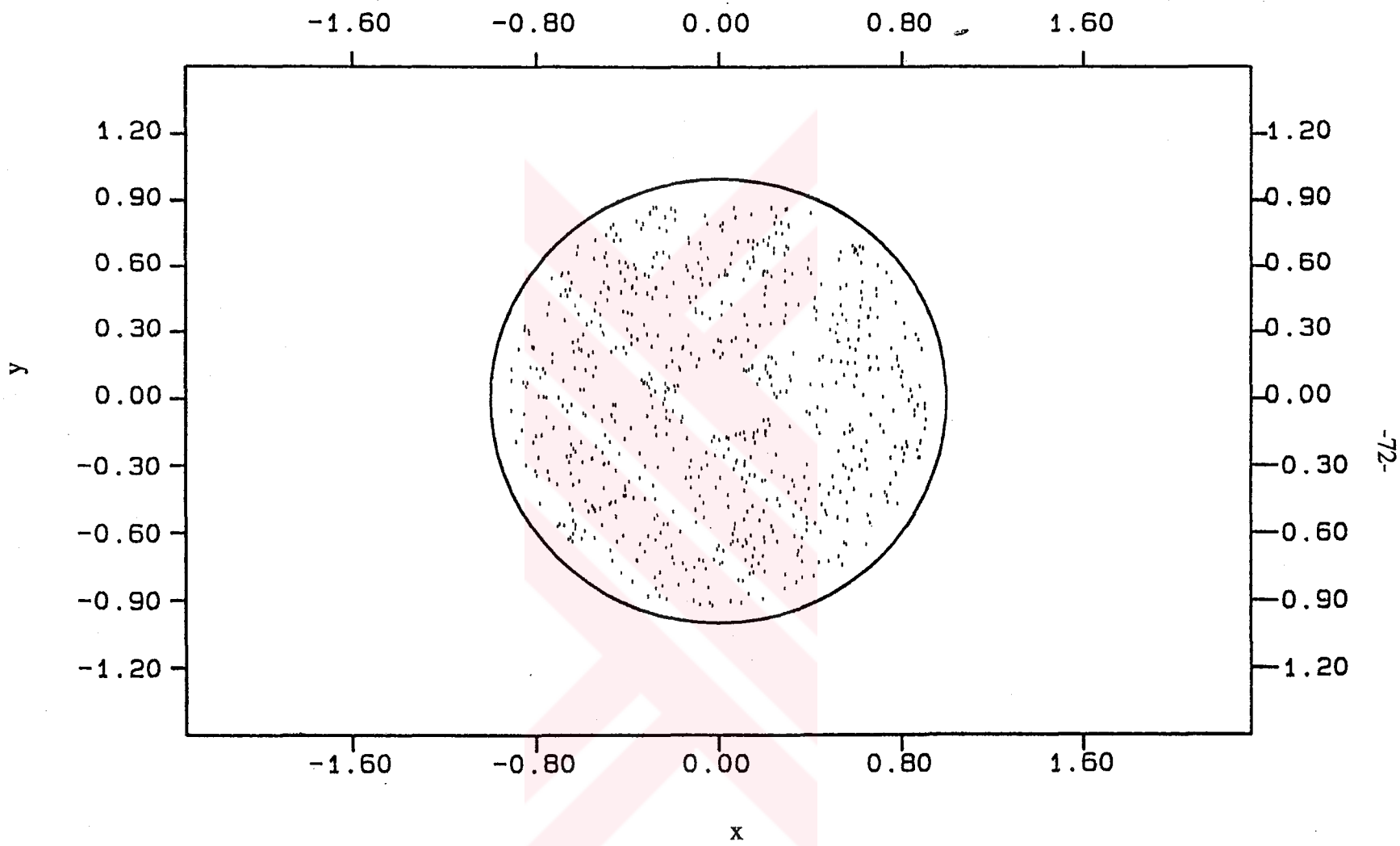


Fig. (5.36) Circular, three vortices on a line with different signs, Poincaré map

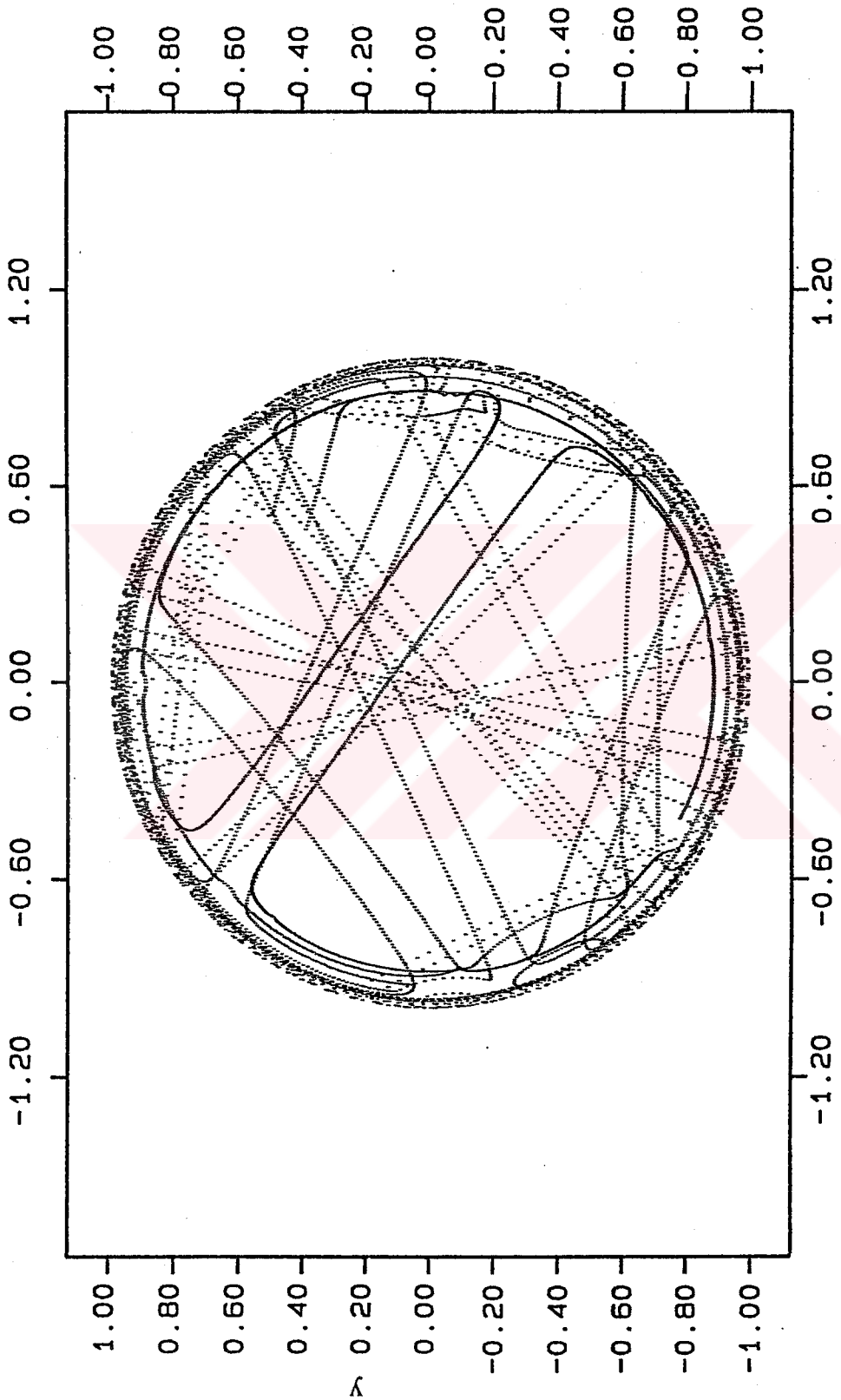


Fig. (5.37) Circular, billiard type chaos, trajectories

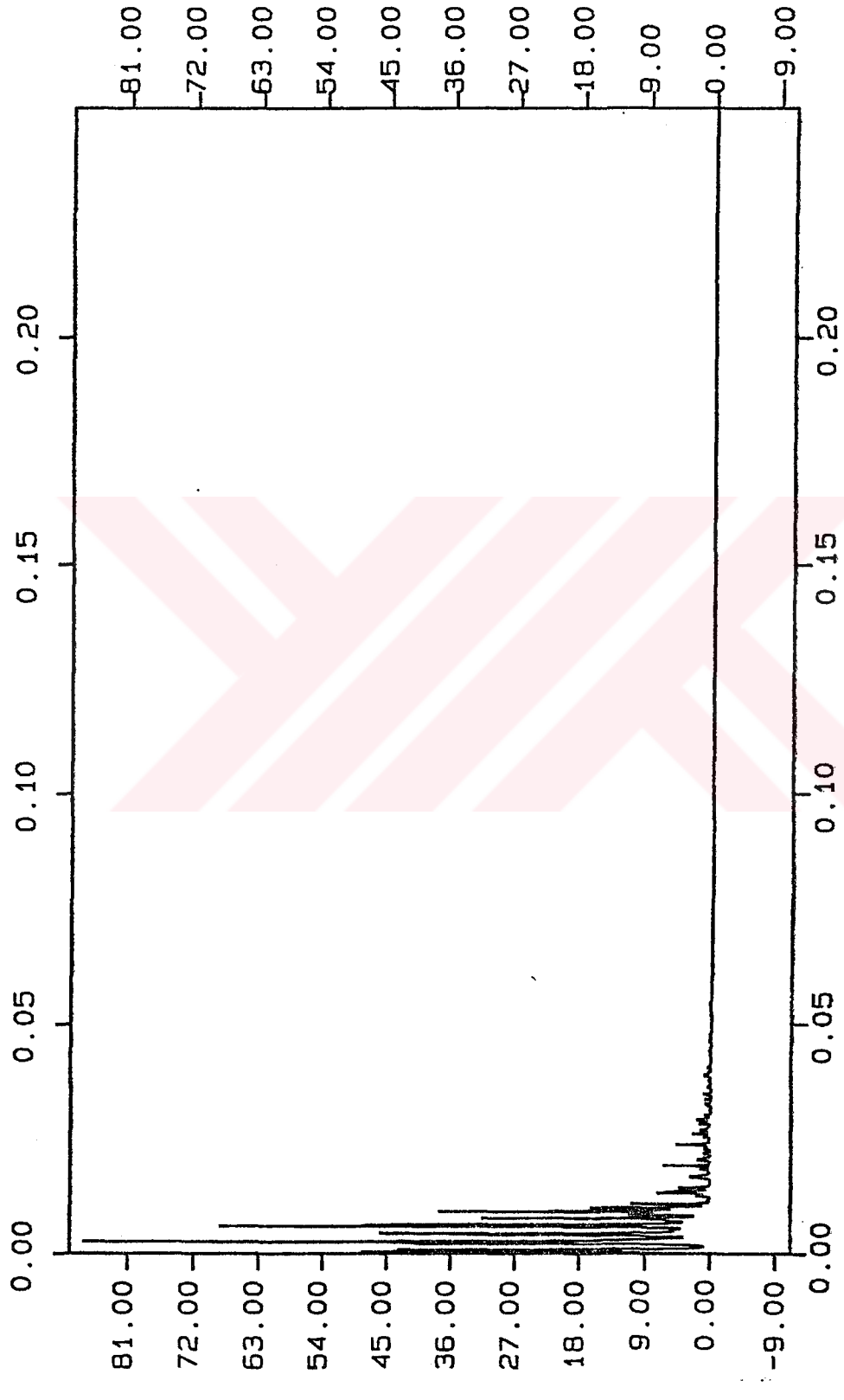


Fig. (5.38) Circular, billiard type chaos, power spectrum

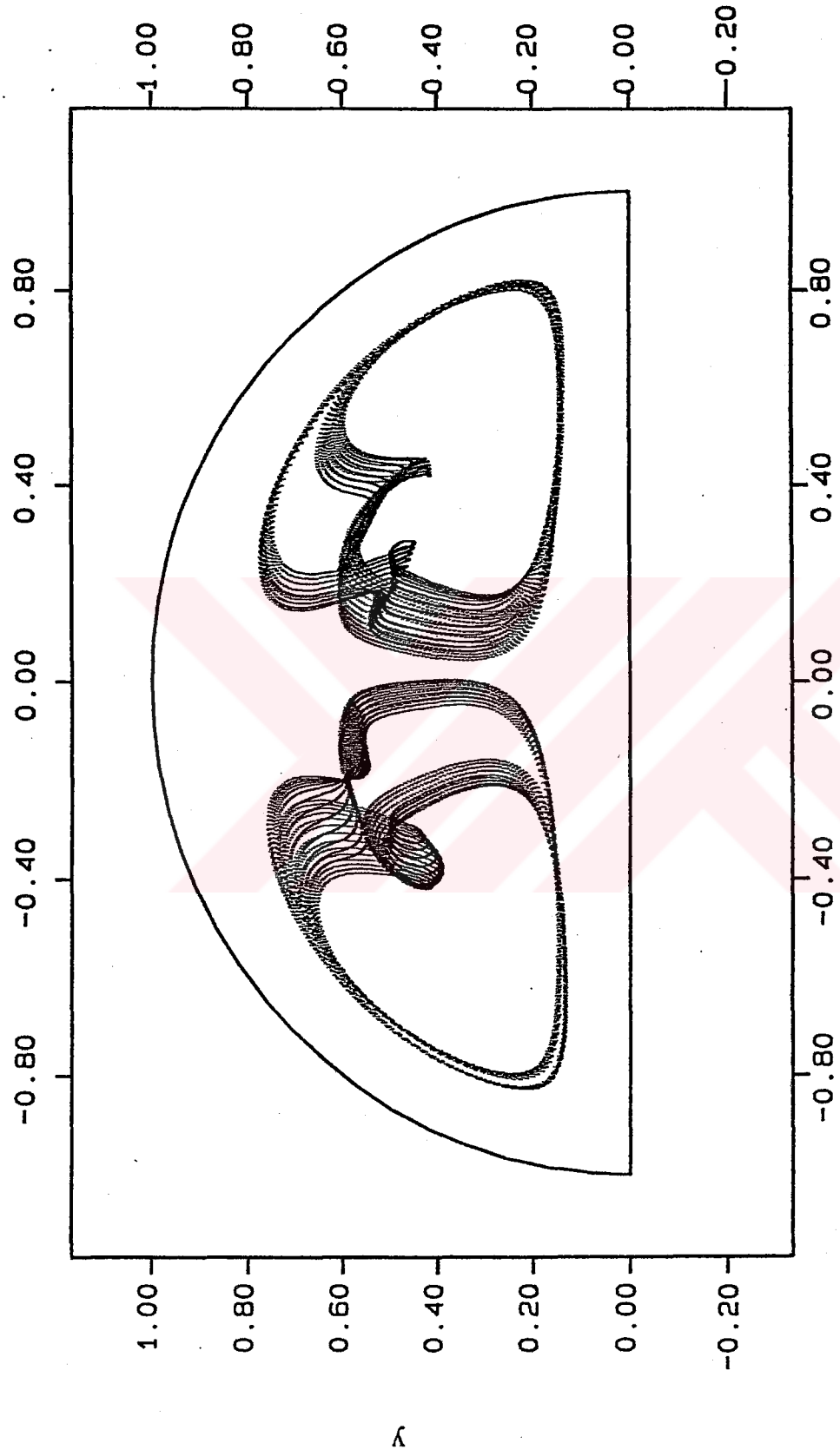


Fig. (5.39) Semi-circular, quasi-periodic motion of two vortices, trajectory

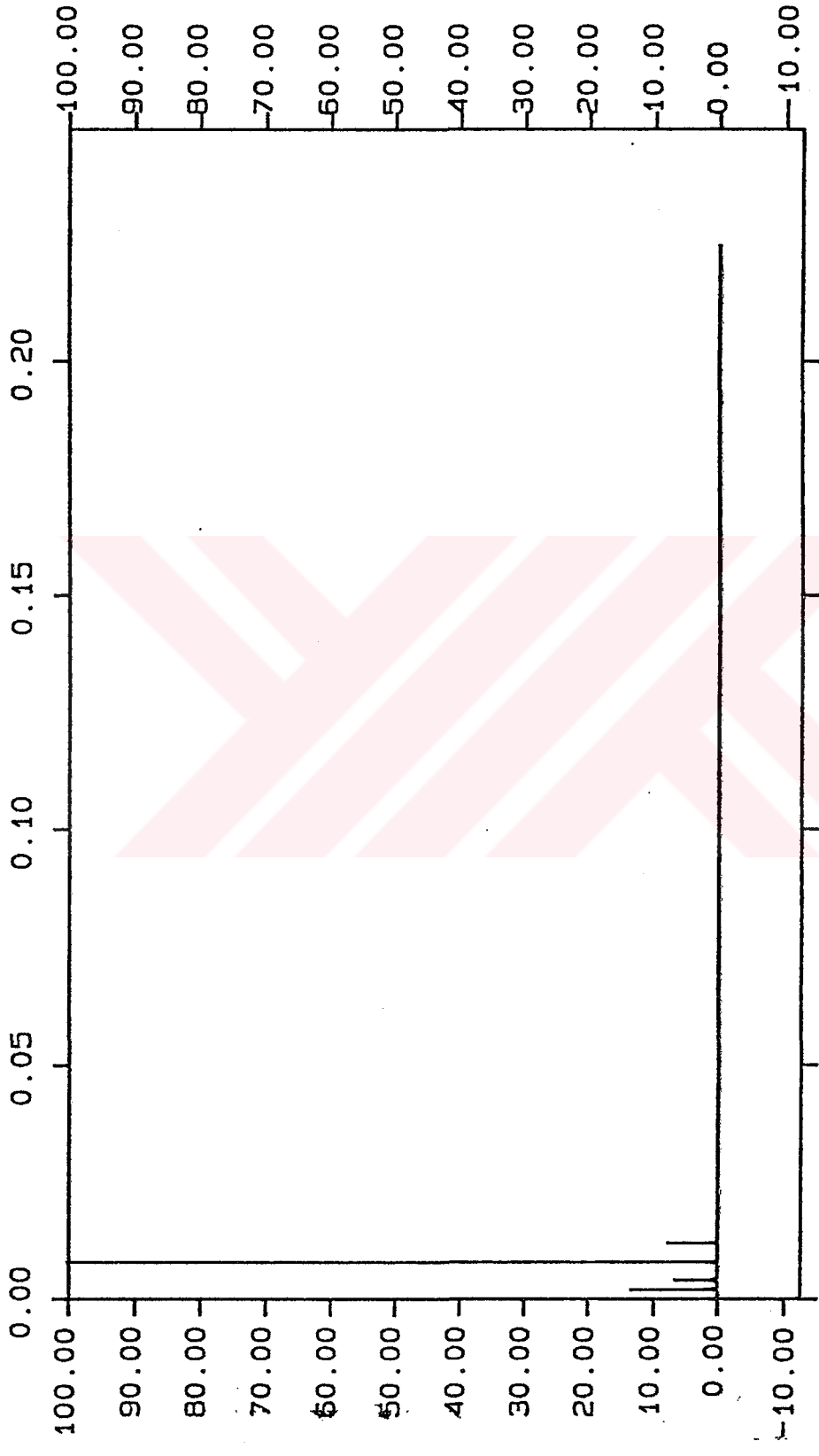


Fig. (5.40) Semi-circular, quasi-periodic motion of two vortices, power spectrum

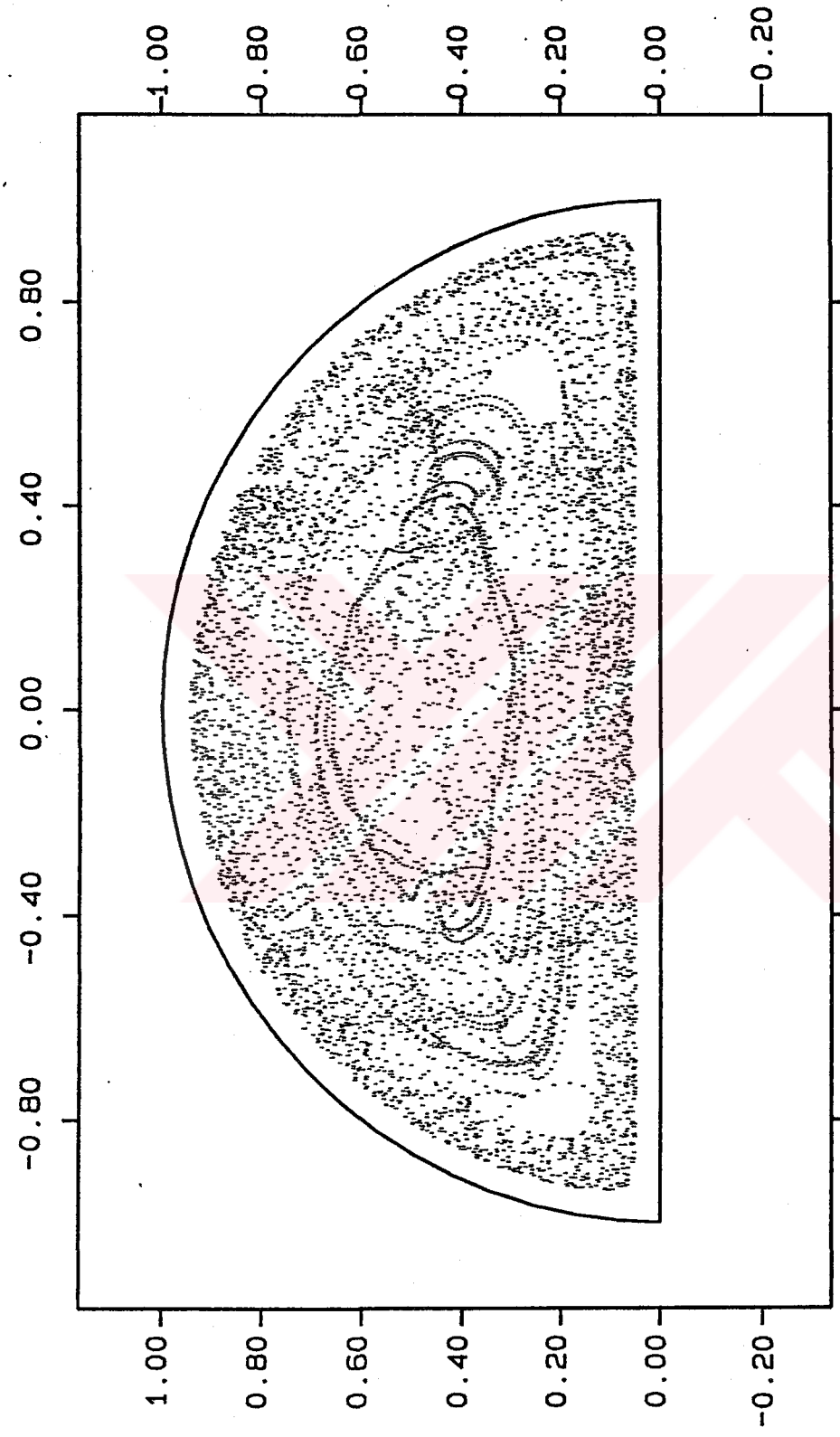


Fig. (5.41) Semi-circular, chaotic motion of two vortices, trajectory

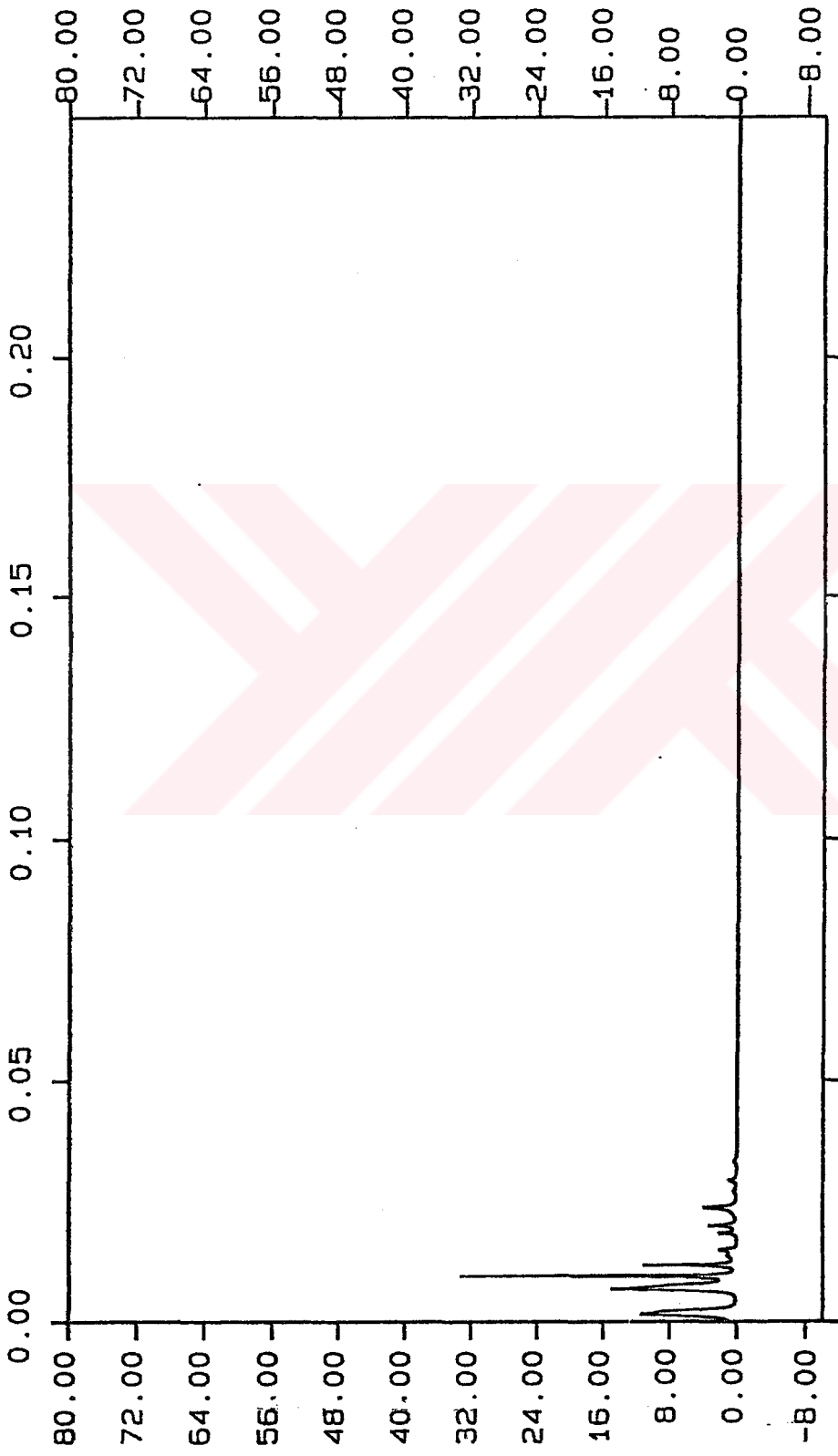


Fig. (5.42) Semi-circular, chaotic motion of two vortices, power spectrum

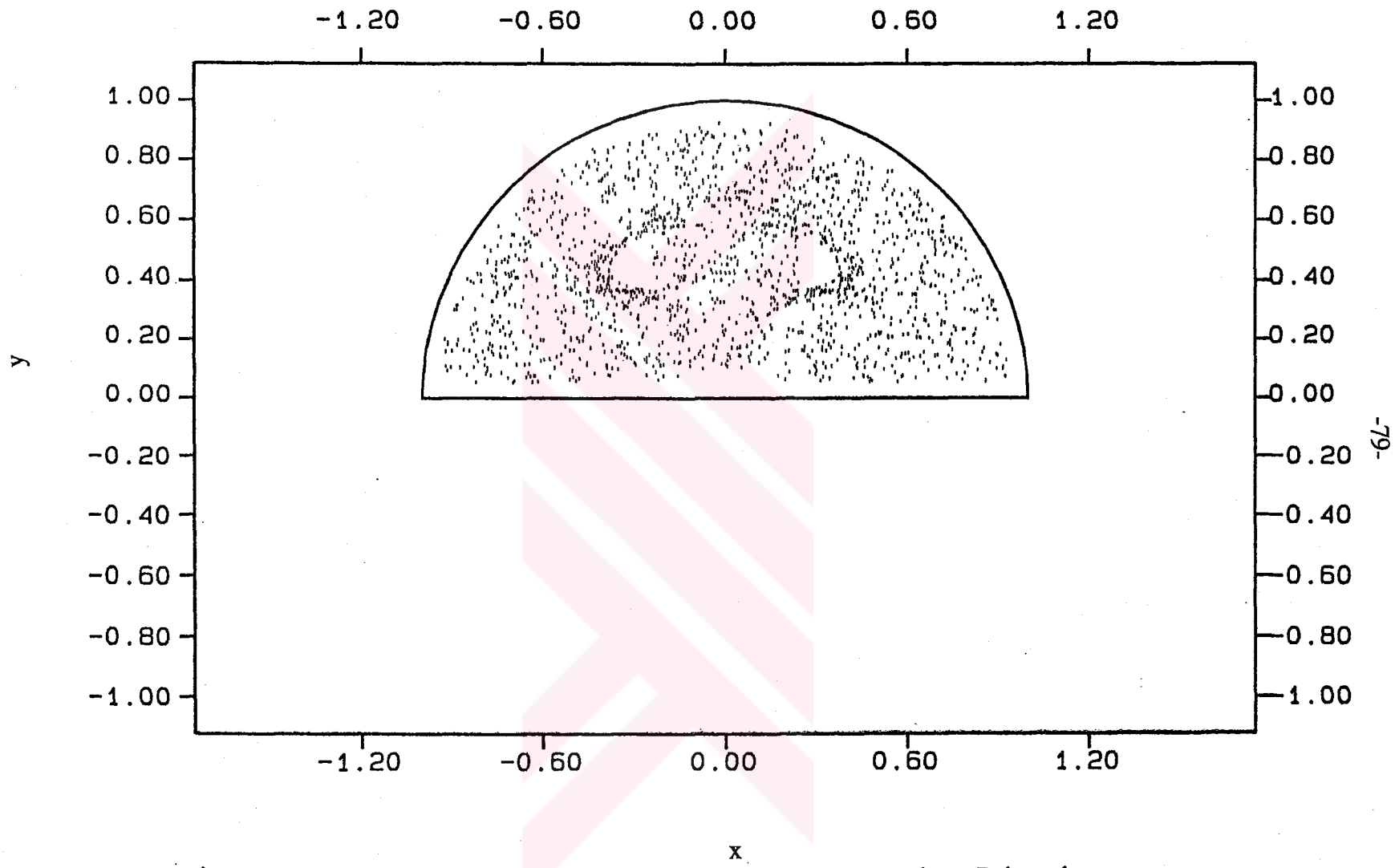


Fig. (5.43) Semi-circular, chaotic motion of two vortices, Poincaré map

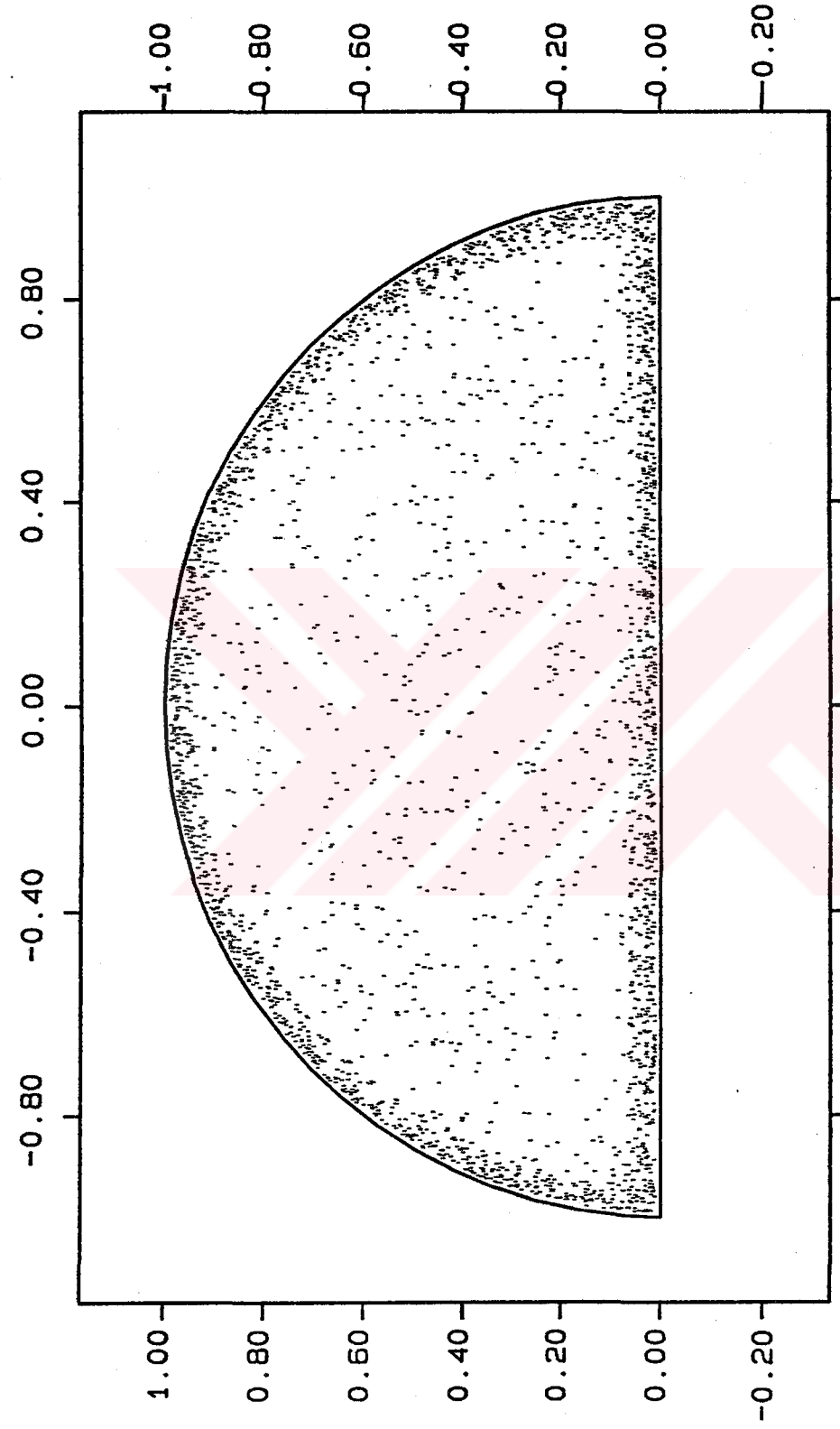


Fig. (5.44) Semi-circular, billiard type chaotic motion of two vortices, trajectory

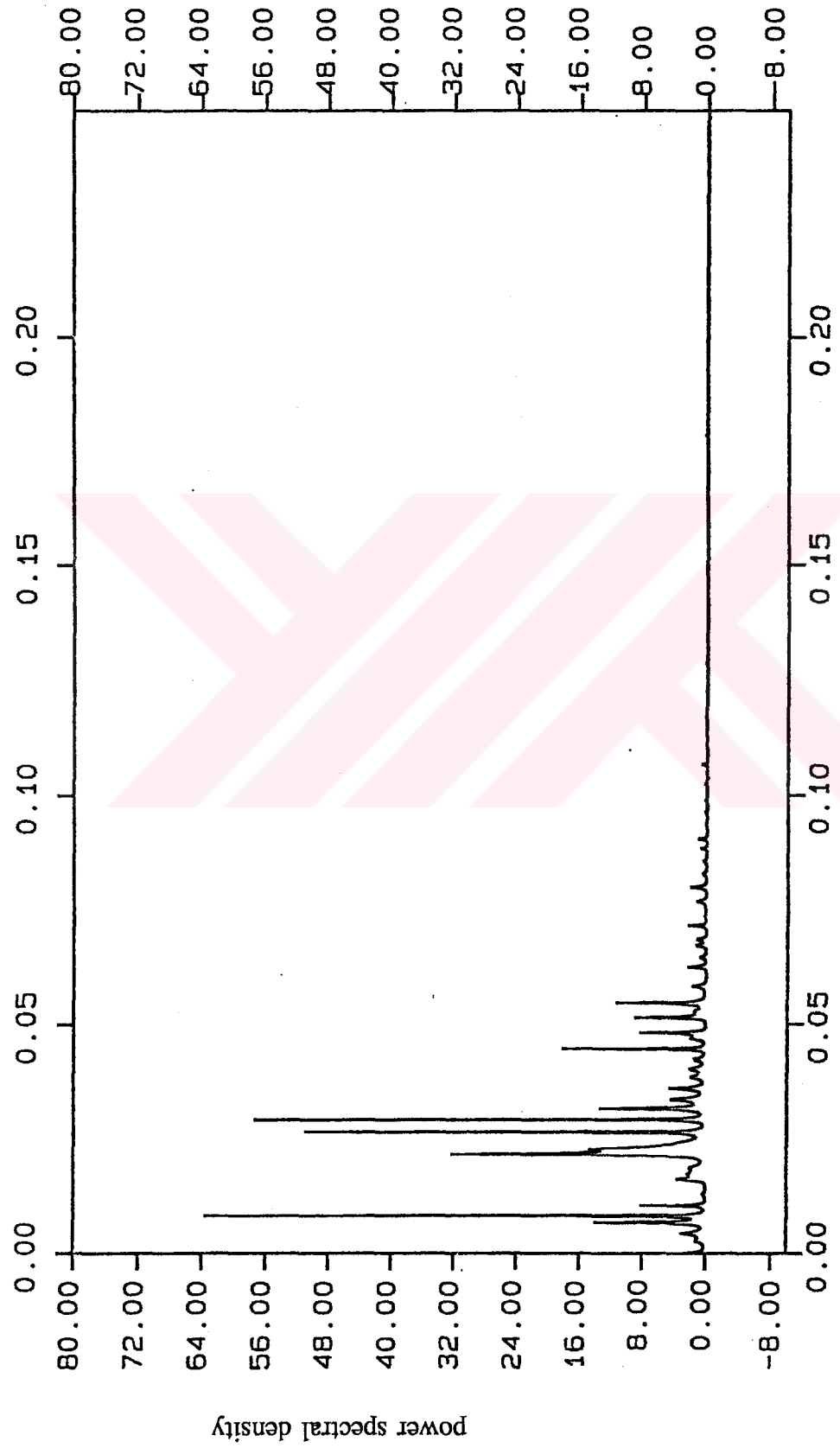


Fig. (5.45) Semi-circular, billiard type chaotic motion of two vortices, power spectrum

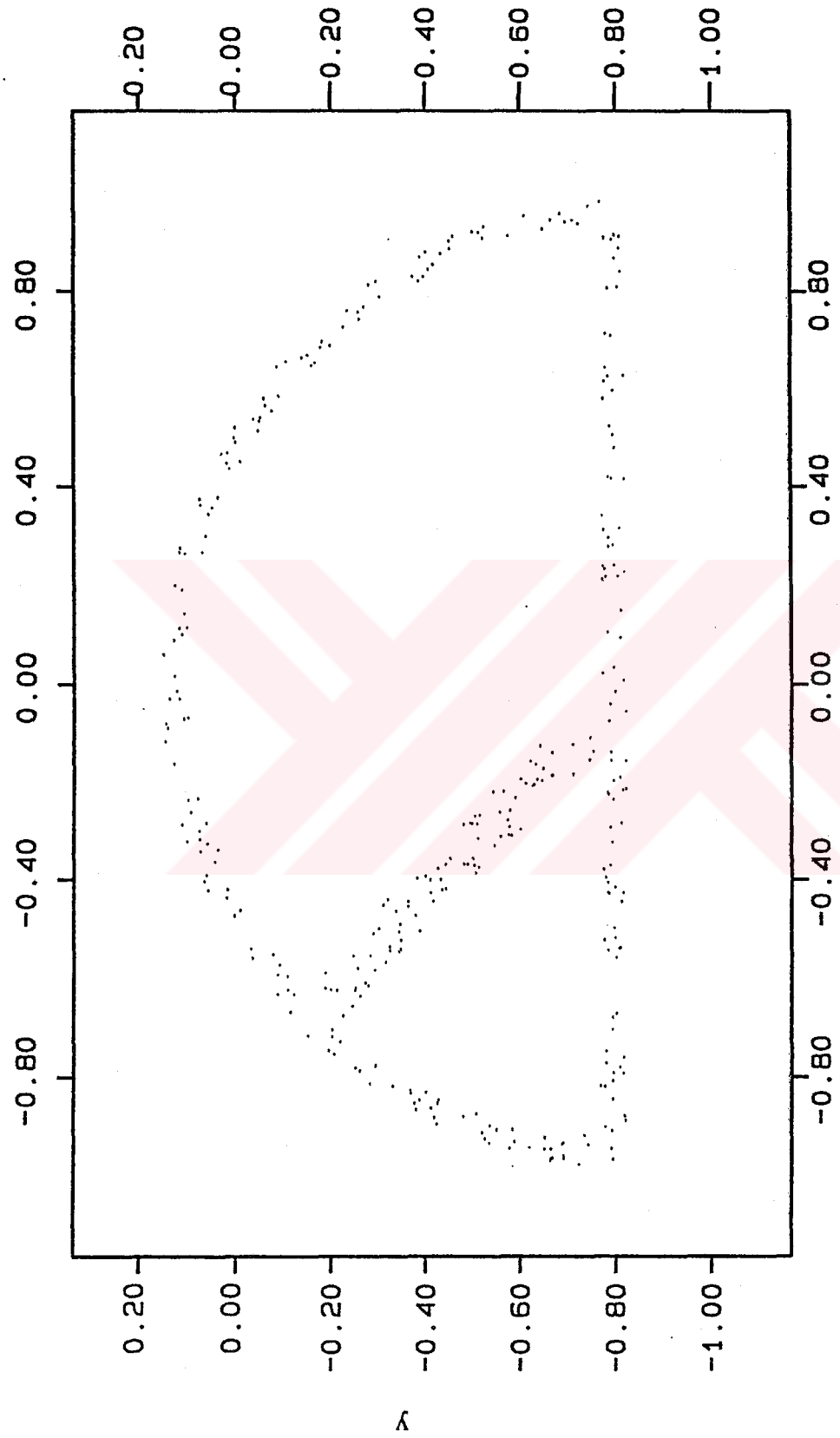


Fig. (5.46) Semi-circular, billiard type chaotic motion of two vortices, Poincaré map

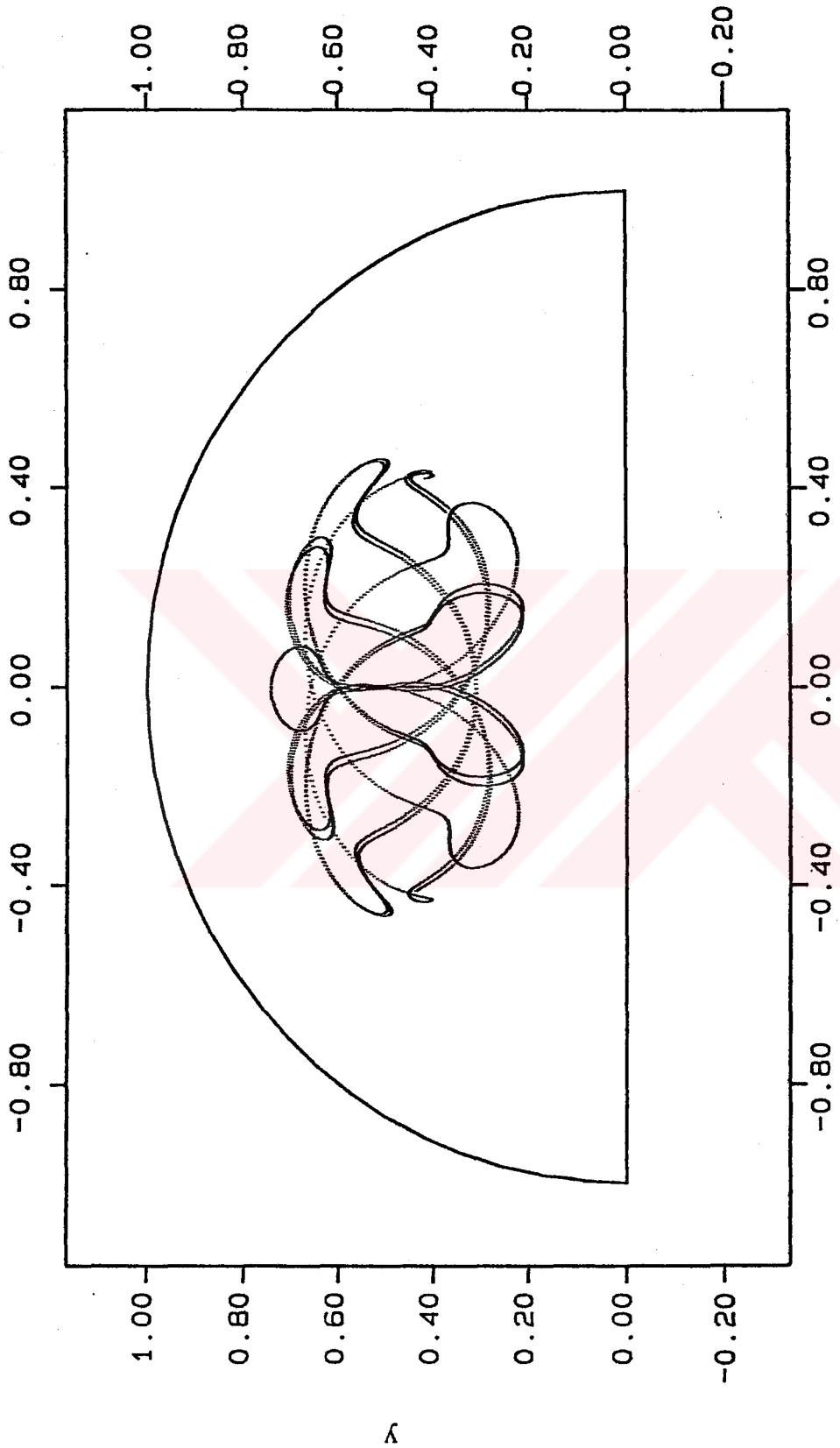


Fig. (5.47) Semi-circular, quasi-periodic motion of two positive vortices, trajectory

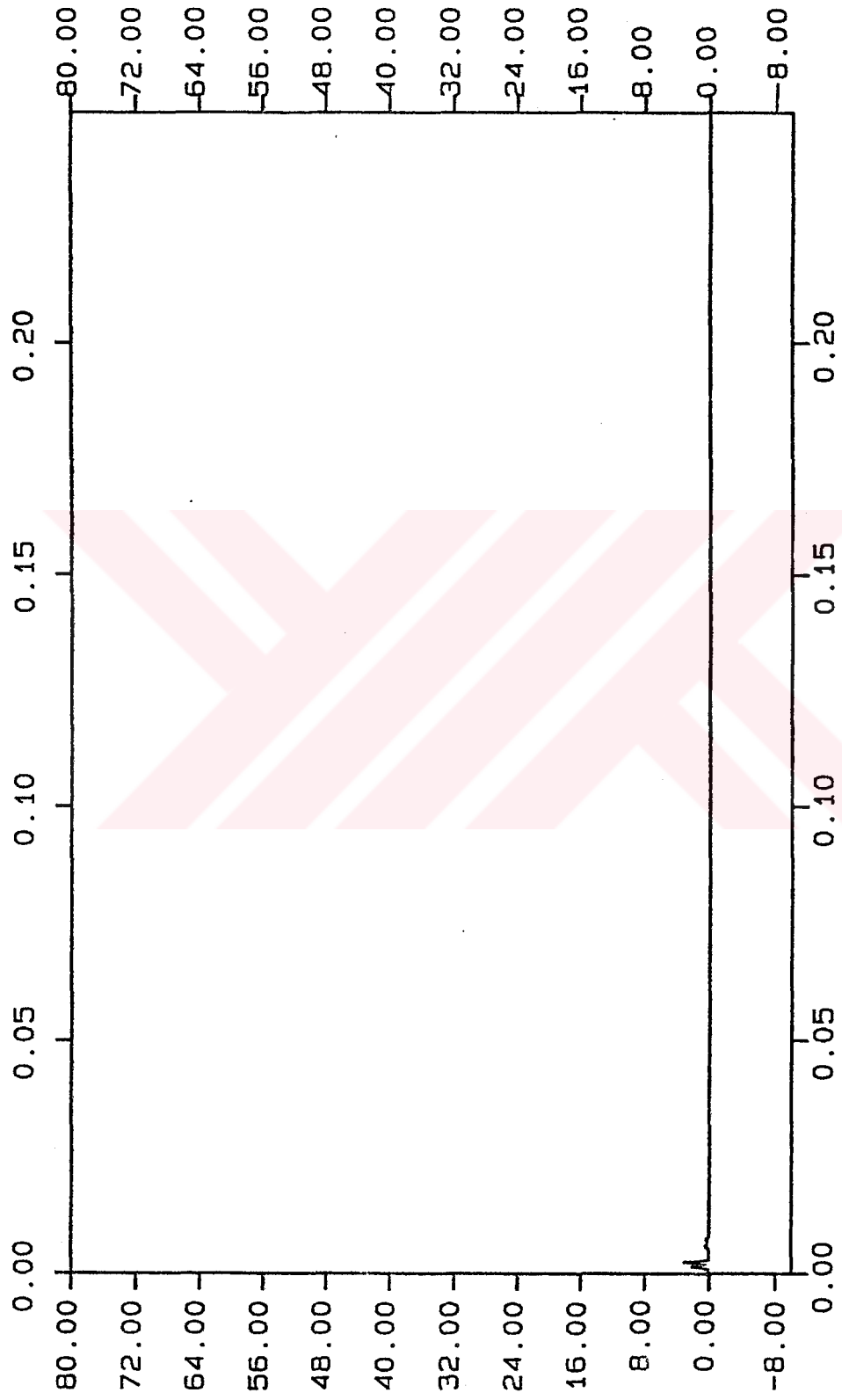


Fig. (5.48) Semi-circular, quasi-periodic motion of two positive vortices, power spectrum

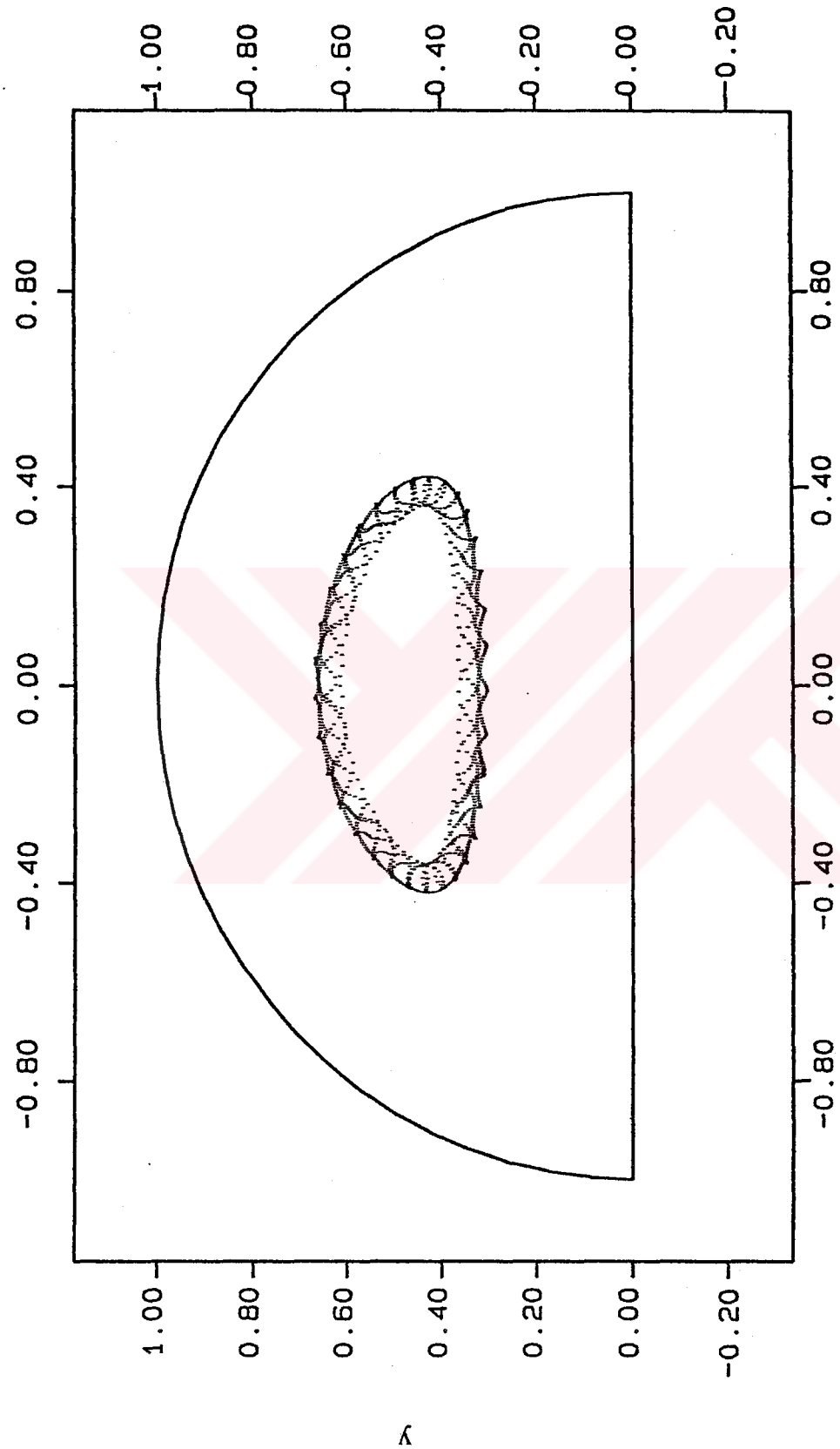


Fig. (5.49) Semi-circular, quasi-periodic motion of two positive vortices, trajectory

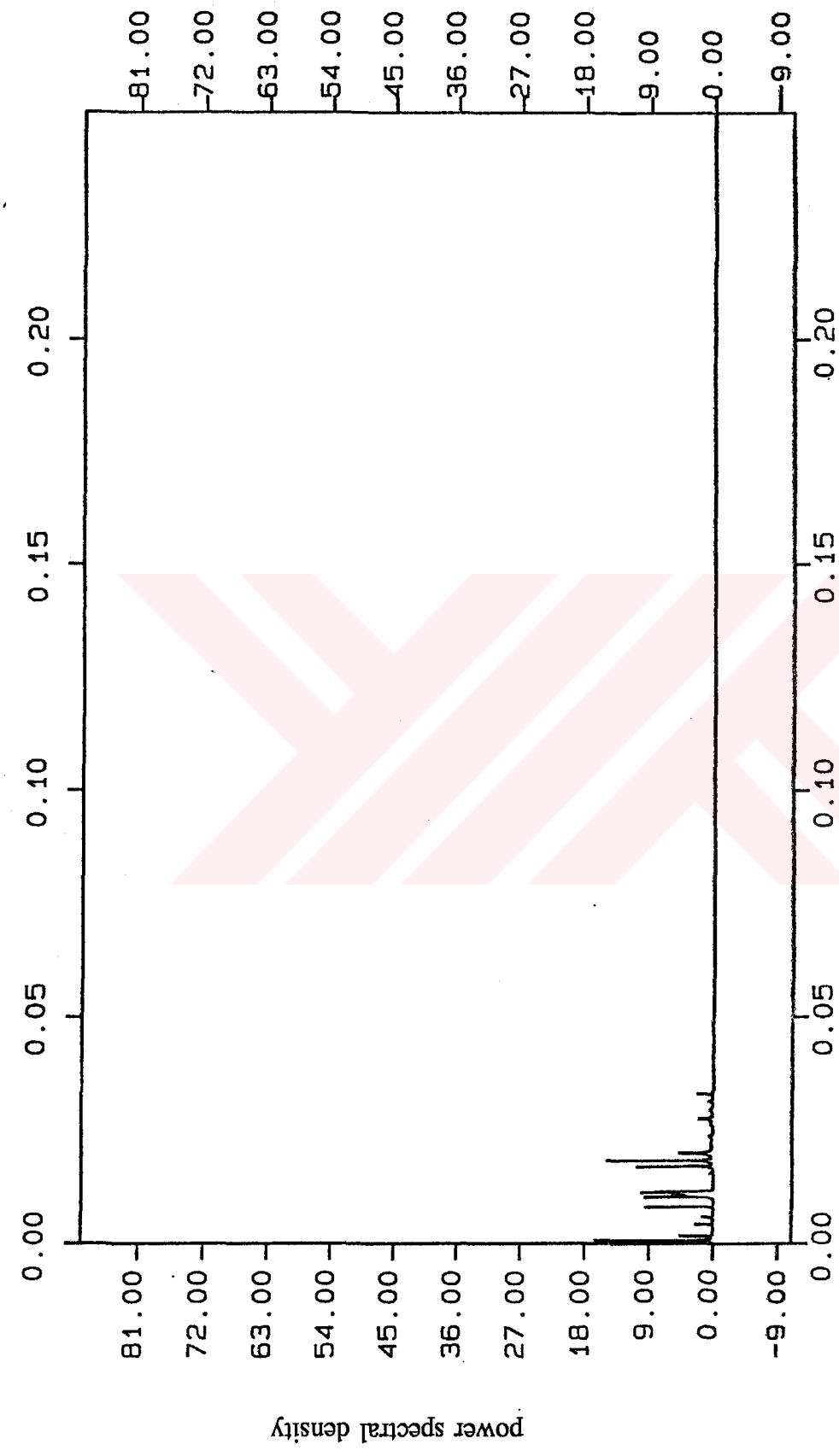


Fig. (5.50) Semi-circular, quasi-periodic motion of two positive vortices, power spectrum

CHAPTER 6
STATISTICAL MECHANICS OF THE POINT VORTICES

In this Chapter, inlight of the numerical results obtained in the previous Chapter, statistical mechanics formulation of point vortices will be undertaken. After introduction, the well-known Onsager paradox will be investigated. Finally, ergodicity of point vortex systems with finite degrees of freedom will be discussed with numerical experiments.

6.1. Background

There is a connection between the theory of dynamical systems and the methods and terminology that have been used up to here. Furthermore there is more interesting connection with the theory of two-dimensional turbulence (Kraichnan and Chen, 1989, and references therein). In a 'plausible' definition of turbulence (Aref, 1990, 1991) includes all stochastic solutions of the equations of motion of two-dimensional hydrodynamics. As it has been seen, in unbounded domain four, in circular domain three, and in the semi-circular domain, two point vortices suffice to produce such stochastic solutions. On the other hand, it may be more precise to describe the stochasticity in above mentioned motions with minimum number of vortices as 'transitions' to statistical mechanics as the number of degrees of freedom increased. Stochastic motions were observed in both dissipative (Lorenz, 1963) and Hamiltonian (Henon-Heiles, 1964) systems with a few degrees of freedom. However, a sound theoretical basis for the statistical mechanics for a few degrees of freedom was absent until 1988 (Berdichevsky, 1988). Thus, the number of works on statistical mechanics of small dimensional chaos is inevitably restricted to a few (von Alberti, 1990).

Statistical mechanics is an asymptotical theory valid in the limit of infinite number of degrees of freedom (Penrose, 1979). Study of small-dimensional chaos starting from the papers by Lorenz (1963) and Henon-Heiles (1964) raises the following question: What is the fundamental hypothesis for the laws of statistical mechanics to be true --- a large number of degrees of freedom or ergodicity?

For ergodic Hamiltonian systems a formulation was suggested by Berdichevsky (1988) stating that ergodicity provides the validity of the laws of equilibrium thermodynamics and statistical mechanics provided that a slight modification is done for finite number of degrees of freedom. However, most small dimensional Hamiltonian systems encountered in physics are not ergodic. Nevertheless it seems plausible that statistical mechanics could be applied to describe systems of such kind if the motion is 'chaotic enough.' One of our aims is to test this hypothesis for the case of unbounded and bounded point vortices. It has already been shown that the motion of the Henon-Heiles oscillators matches very accurately (*i.e.*, with a small percentage of error) the predictions of small-dimensional statistical mechanics for high energy vibrations and the changes that occur when the energy level decreases is alos studied (von Alberti, 1990).

In what follows, a statistical mechanics methodology will be outlined and well known Onsager paradox (Onsager, 1949) will be investigated.

6.2. The Onsager Paradox

The era of studying thermodynamics of vortices was started with Onsager (1949), where he applied the methods of statistical mechanics to describe vortex motion in two-dimensional flow. This formulation emanates from the Eq. (2.3) which is rewritten here for convenience

$$\kappa_j \dot{x}_j = \frac{\partial H}{\partial y_j} , \qquad \kappa_j \dot{y}_j = - \frac{\partial H}{\partial x_j} \tag{6.1}$$

Here x and y are Cartesian coordinates of the point vortex, either in unbounded domain

or in a domain D and κ is its circulation.

Statistical mechanics prescribes the following recipe for calculation of thermodynamical functions for an N-dimensional system with the Hamiltonian $H(p,q)$ (in limit $N \to \infty$). Let $\Gamma(e)$ be the volume of the phase space bounded by the energy hypersurface $H(p,q)=E$,

$$\Gamma(E) = \int_{H(q,p)\le H} dp\,dq. \tag{6.2}$$

Then the entropy $S(E)$ and the temperature T are defined as (Aref, 1991)

$$S(E) = \ln(\frac{d\Gamma}{dE}) + const, \quad \frac{1}{T} = \frac{dS(E)}{dE} = \frac{d\Gamma}{dE} / \frac{d^2\Gamma}{dE^2}. \tag{6.3}$$

Via this methodology, Onsager (1949) discovered that temperature may be negative for a vortex motion in a container. By Eq.(6.2), the volume $\Gamma(E)$ increases monotonically with increasing energy E. When $E \to \infty$, any position in the container becomes admissible for each vortex. Therefore, $\Gamma(E) \to A^N$ for $E \to +\infty$, here A is the container area and N is the number vortices. Thus, the function $\Gamma(E)$ tends to a constant as $E \to \infty$, and $d\Gamma/dE$ tends to zero. From the relation

$$0 < \frac{d\Gamma}{dE} = -\int_E^\infty d^2\Gamma/dE^2\,dE, \tag{6.4}$$

it follows that $d^2\Gamma/dE^2$ must be negative for some values of E, and, therefore, the temperature given by Eq. (6.3) is negative for these values of E. However, a theorem proven by Frolich and Ruelle (see, Eckmann and Ruelle, 1985) states that the negative temperature disappears in the limit $N \to \infty$. More than this, based upon the developments by Berdichevsky (1988), Berdichevsky, Kunin, and Hussain (1991) put forward the following remedy.

6.3. Point Vortices As a Finite Degree of Freedom System

For ergodic Hamiltonian systems the equipartition law (energy is shared equally among all degrees of freedom) is valid (Walker and Ford, 1969)

$$\langle p_1 \frac{\partial H}{\partial p_1} \rangle = \cdots = \langle p_N \frac{\partial H}{\partial p_N} \rangle = T \tag{6.5}$$

where $<.>$ denotes the time average along the trajectory and the common value T is called 'temperature.' In accordance with Birkhoff's theorem, the time average depends on the energy surface only, and does not depend on a selected trajectory. To obtain a physical meaning of the temperature T for point vortices, we refer to the equipartition law Eq. (6.6), which, in the case of vortex dynamics, takes the form

$$T = \kappa_1 \langle y_1, \dot{x}_1 \rangle = \cdots = \kappa_N \langle y_N \dot{x}_N \rangle . \tag{6.6}$$

The quantity $<y_1 \dot{x}_1>$ denotes the average area, bounded by the first vortex trajectory, related to the time unit:

$$\langle y_1 \dot{x}_1 \rangle = \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^\tau y(t)\dot{x}(t)\,dt = \lim_{t \to \infty} \frac{1}{\tau} \int_0^\tau y\,dx . \tag{6.7}$$

Hence the temperature in the vortex dynamics (in contrast to statistical mechanics) represents a geometrical characteristic of trajectories. The dimension of the temperature id the usual one and it is equal to the dimension of the kinetic energy per unit area. The equipartition law, applied to vortex dynamics, means that the products of mean areas and circulations are the same for all vortices.

Since T is defined as the common value of the partition law Eq. (6.6) and S, T, and Γ are given as follows (Berdichevsky, 1988)

$$S = \ln\Gamma + const, \qquad T = \Gamma/\frac{\partial\Gamma}{\partial E} \tag{6.8}$$

Expressions Eq. (6.3) and Eq. (6.7) for entropy were pointed out by Gibbs (1902), and it is essential that they coincide in the limit $N \to \infty$. Historically, formula Eq. (6.3) was emphasized in the literature and the equivalent second expression almost never mentioned. We will now observe how important this difference is. The vortices of the positive (negative) sign move, on the average, clockwise (counterclockwise), but T always remains positive. The temperature T increases with energy (for large enough energy) because $d\Gamma/dE \to 0$ for $E \to \infty$, and it follows from Eq. (6.7) that $T \to +\infty$ (Berdichevsky, Kunin, and Hussain, 1991).

With regard to above analysis, therefore, the works of Montgomery and Joyce (1974) and, especially reflections of Aref (1991) on this matter do contain an error. We now confine more attention on the ergodicity of point vortices in order to apply the methodology of statistical mechanics.

6.4. Ergodicity of the Point Vortices

Numerical investigation of ergodicity has been started with Fermi, Pasta, and Ulam (see, Fermi, 1965), where the authors studied the equipartition law for chain of nonlinear oscillators. Recently, several researchers (Thirumalai and Mountain, 1989 and Pettini and Landolfi, 1990, see Weiss and McWilliams, 1991) have tested ergodicity in plasma dynamics (see, . They have also tried to show that equipartition be reached after "enough time" was spent. Thus, the following question comes into mind: Is there any energy threshold above which the motion can be ergodic? For Henon-Heiles (1964), for instance, this threshold corresponds to the maximum energy level of 1/6, that is the 'escape' energy.

Works on point vortex systems in unbounded domain (Novikov and Sedov, 1979, Aref and Pomphrey, 1982, Eckhart and Aref, 1988) did not make conclusive

comments on ergodicity. On the other hand, Weiss and Mc Williams (1991) showed that the dynamics of point vortices in a two-dimensional square doubly periodic domain is not ergodic. By comparing the time average with ensemble average, they have found that the nonergodicity is not due to a gross fragmentation of phase space which usually comes from a broken symmetry. However, they have postulated that there might be one region covering a large part of the surface and one or more much smaller regions in the remainder. Therefore, one cannot rule out the possibility that, rather than being truly separate components, the regions are separated by weakly permeable barriers and hence the dynamics 'ultimately' ergodic.

In the unbounded domain four identical vortices placed in a equi-lateral face-centered triangle with the center vortex perturbed 0.2 unit in both x and y axis when the side of the triangle is 1 unit is tested for equipartition.

After 10820 units of time with 0.01 time step, averages of the temperatures of the point vortices were 0.119358 and maximum deviation from the mean average was 1.9%. The evolution of the temperatures is found to be converging (Fig. (6.1)). Therefore this system is ergodic.

The billiard type chaos is proven to be ergodic by Sinai (1970, 1972). Therefore, billiard type motions found in circular and semi-circular domain resembles elastic collision of hard spheres and thus they are ergodic.

Probability distribution functions for ergodic motions found in few vortex systems in unbounded and bounded domains are being calculated both theoretically and experimentally.

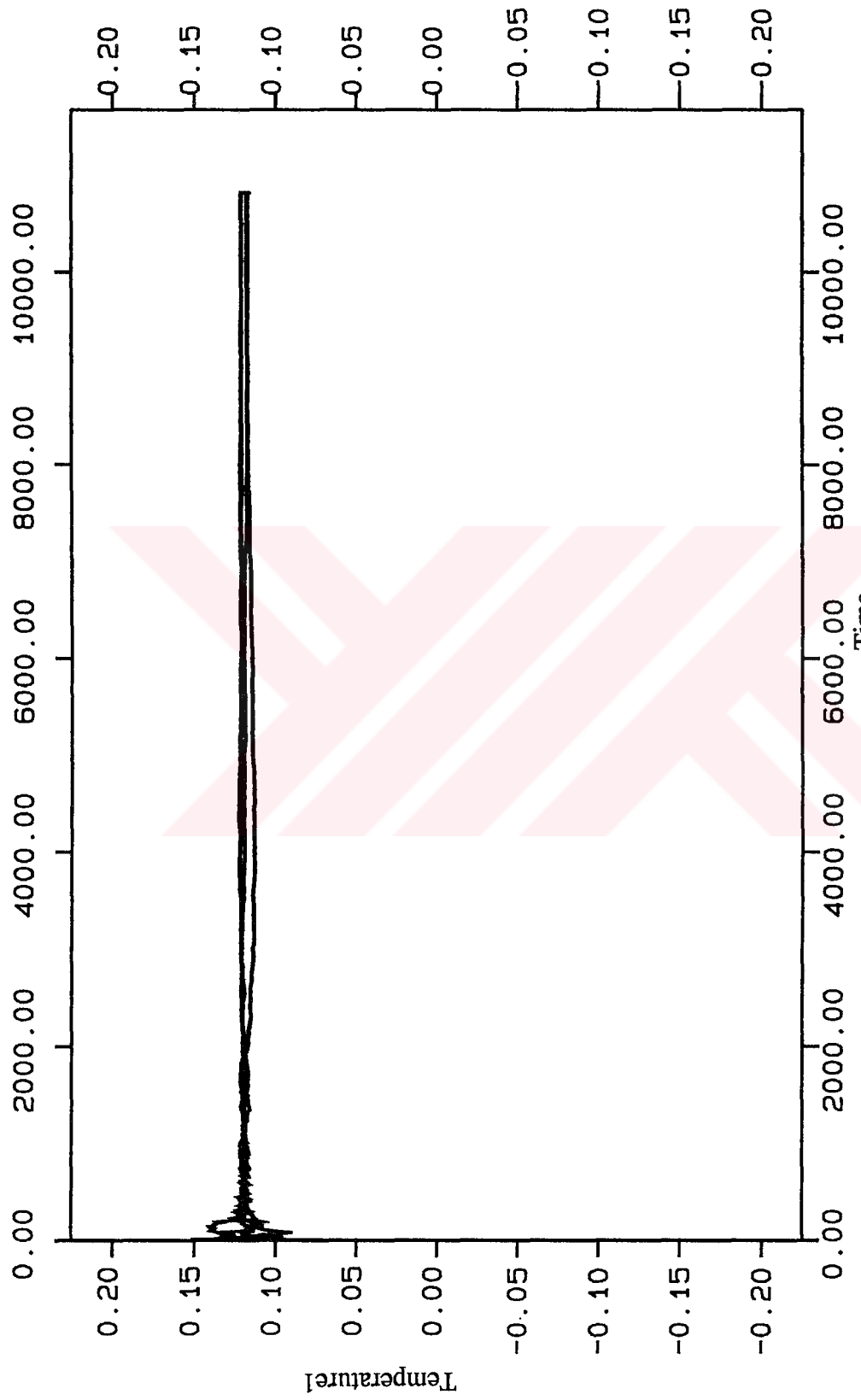


Fig. (6.1) Temperatures of four vortices in an unbounded domain

CONCLUDING REMARKS

Chaotic and regular motions are observed in unbounded, circular, and semi-circular domains by drawing trajectories, power spectrums, and Poincaré maps. Investigations of the characteristics of these fixed points showed that both stable and unstable fixed points exist.

For the unbounded case with two vortices, it is found that there is no fixed point of the system. While three vortices give an unstable fixed solution, four vortices has a stable fix solution. It is known that four vortices in unbounded domain could give chaotic solutions depending the configuration of initial conditions. Symmetries yield quasi-periodic behavior. For the equilateral face-centered triangle small perturbations of the initial conditions gives chaotic behavior. In the rectangular case it is found that small perturbations does not makes the system chaotic. In this case trajectories continue to remain on the tori. However large perturbations can make the system to behave chaotic.

It is found that in a line of vortices placed symmetrically in a configuration of (-1, -0.5, 0.5, 1) with different pairs of strengths shows an interesting phenomena. When vortices of bigger strength are placed inside, starting with a quasi-periodic motion, system becomes chaotic after sometime. However, other configurations of strengths does not give chaotic motion. This phenomena could not be explained with the constants in involution since all cases has similar constants in involution.

Introducing boundaries to the system reduces the threshold to observe chaotic behavior. In circular domain it is found that three vortices is enough to make the system chaotic. In a semi-circular domain two vortices are investigated and their fixed points are found to be stable. As this fixed solutions perturbed more chaotic behavior

is observed.

For the unbounded case of four vortices equipartition of the system is tested and it is found that the system is ergodic. Therefore it would be possible to use statistical mechanics for chaotic point vortex motion. This would help to understand the global characteristics of the system.

A new approach to numerical analysis is introduced using object oriented programming. This technique with its modularity and simplicity proved that it deserves more attention. However both Runge-Kutta and Bulirsch-Stoer methods for numeric integration of ODEs found to be slow. New techniques for fast and accurate integration is being studied (Bursal and Tongue, 1992).

Lyapunov exponents given for the point vortices is not encountered in the literature. Trying to estimate Lyapunov exponents is still an ongoing process. New algorithms and analytical simplifications will be tried. Studying fractal structure of the system and calculating the temperature and other physical characteristics analytically are the future works planned to be done.

Trying to obtain a chaotic behavior with one point vortex is also being investigated. Recent experiments with quarter of a circle failed to give chaotic motion with one point vortex (these experiments are not shown in this work). Quarter of a ellipse will be investigated to obtain positive answer.

REFERENCES

ALBERTI, M., v., (1990), Chaotic Motion of Henon-Heiles Oscillators, M.S. Thesis, Georgia Inst. of Tech., September.

AREF, H., (1983), Integrable, Chaotic, and Turbulent Vortex Motion in Two-Dimensional Flows, Annu. Rev. Fluid Mech., 15, pp. 345-389.

AREF, H., (1984), Stirring by Chaotic Advection, J. Fluid Mech., 143, pp. 1-21.

AREF, H., (1986), The Numerical Experiment in Fluid Mechanics, J. Fluid Mech., 173, pp. 15-41.

AREF, H., (1990), Chaotic Advection of Fluid Particles, Phil. Trans. Roy. Soc. Lond. A, 333, pp. 273-288.

AREF, H., (1991), Stochastic Particle Motion in Laminar Flows, Phys. Fluids A, 3 (5), pp. 1009-1016.

AREF, H., and KAMBE, T., (1988), Report on the IUTAM Symposium: Fundamental Aspects of Vortex Motion, J. Fluid Mech., 190, pp. 571-595.

AREF, H., JONES, S. W., MOFINA, S., and ZAWADZKI, I., (1989), Vortices, Kinematics and Chaos, Physica D, 37, pp. 423-440.

AREF, H., and POMPHREY, N., (1980), Integrable and Chaotic Motions of Four Vortices, Phys. Lett., 78A (4) pp. 297-301.

AREF, H., and POMPHREY, N., (1982), Integrable and Chaotic Motions of Four Vortices. The Case of Identical Vortices, Proc. Roy. Soc. Lond. A, 380, pp. 359-387.

AREF, H., ROTT, N., and THOMANN, H., (1992), Gröbli's Solution of the Three-Vortex Problem, Annu. Rev. Fluid Mech., 24, pp. 1-20.

BERDICHEVSKY, V. L., (1988), The Connection Between Thermodynamic Entropy and Probability, Prikl. Matem. Mekhan., 52 (6), pp. 738 -- 746.

BERDICHEVSKY, V. L., KUNIN I., and HUSSAIN F., (1991), Negative Temparature of Vortex Motion, Phys. Rev. A, 43 (4), pp. 2050-2051.

BURSAL, E. H., and TONGUE, B. H., (1992), A Hybrid Symbolic-Numerical Method

for Integrating Ordinary Differential Equations, J. Sound Vibration, 26 (3), pp. 295-304.

CARRIER, G. F., KROOK, M., and PEARSON, C. E., (1966), Functions of a Complex Variable, Theory and Technique, McGraw-Hill, pp. 111-182.

ECKHARDT, B., (1988), Integrable Four Vortex Motion, Phys. Fluids, 31, pp. 2796-2801.

ECKHARDT, B., and AREF, H., (1988), Integrable and Chaotic Motions of Four Vortices II. Collision Dynamics of Vortex Pairs, Phi. Trans. R. Soc. Lond. A, 326, pp. 655-696.

ECKMANN, J. P., and RUELLE, D. (1985), Ergodic Theory of Chaos and Strange Attractors, Rev. Mod. Phys., 57 (3) Part 1, 617-656. Addendum: Rev. Mod. Phys., 57 (4), p. 1115.

ECKMANN, J. P., and RUELLE, D. (1992), Fundamental Limitations for Estimating Dimensions and Lyapunov Exponents in Dynamical Systems, Physica D, 56, pp. 185-187.

FERMI, E., (1965), Collected Papers, Vol. II, University of Chicago Press, p. 978.

FORD, J., (1974), Stochastic Behavior in Nonlinear Oscillator Systems, Lecture Notes in Physics.

GIBBS, J., (1902), Thermodynamics. Statistical Mechanics, Yale University Press.

GLEICK, J., (1990), Chaos, Making a New Science, Penguin Books, pp. 9-33.

GOLDHIRSCH, I., SULEM, P-L., and ORSZAG, S. A., (1987), Stability and Lyapunov Stability of Dynamical Systems: A Differential Approach and A Numerical Method, Physica D, 27, pp. 311-337.

GURTIN, M. E., (1972), The Linear Theory of Elasticity, in Encyclopedia of Physics, Vol. VIa/2, Mechanics of Solids, Ed. by C. Truesdell, p. 19.

HENON, M., and HEILES C., (1964), The Applicability of the Third Integral of Motion: Some Numerical Experiments, The Astronomical Journal, 69 (1), 73-79.

KAYKAYOĞLU, C. R. and ERERTEM, C., (1987), Vorteks-Cisim Etkileşimi İçin Bir Bilgisayar Benzetişim Modeli, Bilgisayar Benzetişim Yöntemlerinin Fizik ve Mekanik Problemlerine Uygulanması Sempozyumu, pp. 175-220.

KIMURA, Y., KUSUMOTO, Y., and HASIMOTO, H., (1984), Some Particular Solutions for Symmetric Motion of Point Vortices in a Circular Cylinder, J. Phys. Soc.

Japan, 53, pp. 2988-2995.

KIMURA, Y., and HASIMOTO, H., (1986), Motion of a Pair of Vortices in a Semicircular Domain, J. Phys. Soc. Japan, 55, pp. 5-8.

KRAICHNAN, R. H., and CHEN, S., (1989) Is There a Statistical Mechanics of Turbulence?, Physica D, 37, pp. 160-172.

LAMB, H., (1932), Hydrodynamics, 6th edition, Dover.

LORENZ, E. N., (1963), Deterministic Non-Periodic Flow, J. Atmosph. Sci., 20, pp. 130-141.

LUPINI, R., and SIBONI, S., (1991), Chaotic Motions of Inviscid Point Vortices Around Rigid Obstacles, Il Nuovo Cimento, 106, pp. 957-962.

MILNE-THOMSON, L. M., (1958), Theoretical Aerodynamics, Dover Publications.

MONTGOMERY, D., and JOYCE, G., (1974) Statistical Mechanics of "Negative Temperature" States, Phys. Fluids, 17, (6), pp. 1139-1145.

NOVIKOV, E. A., (1976), Dynamics and Statistics of a System of Vortices, Sov. Phys. JETP, 41 (5), pp. 937-943.

NOVIKOV, E. A., and SEDOV, Yu. B., (1978), Stochastic Properties of a Four-Vortex System, Sov. Phys. JETP, 48 (3), pp. 440-444.

NOVIKOV, E. A., and LEDOV, Y. B., (1980), Stochastization of Vortices, JETP Lett., 29 (12), pp. 677-679.

OKAMOTO, H., and KIMURA, Y., (1989), Chaotic Advection by a Point Vortex in a Semidisk, Topological Fluid Mechanics Proceedings of the IUTAM Symposium, pp. 105-113.

ONSAGER, L., (1949), Statistical Mechanics of Discrete Line Vortices, Nuovo Cimento Suppl., 6, pp. 279-293.

PARKER, T., S., and CHUA, L., O., (1989), Practical Numerical Algorithms for Chaotic Systems, Springer-Verlag.

PENROSE, O., (1979), Foundations of Statistical Mechanics, Rep. Prog. Phys., 42, pp. 1937-2006.

PRESS, W., H., (1988), Numerical Recipes in C, Cambridge University Press.

ROBINSON, S. K., (1991), Coherent Motions in the Turbulent Boundary Layer, Annu.

Rev. of Fluid Mech., 23, pp. 601-639.

RUELLE, D., (1989), Chaotic Evolution and Strange Attractors The Statistical Analysis of Time Series for Deterministic Nonlinear Systems, Cambridge University Press.

SAGDEEV, R. Z., USIKOV, D. A., and ZASLAVSKY, G. M., (1988), Nonlinear Physics From Pendulum to Turbulence and Chaos, Harwood Academic Publishers.

SCHILDT, H., (1991), C++: The Complete Reference, Osborne McGraw-Hill.

SINAI, Ya. G., (1970), Dynamical Systems with Elastic Reflections, Sov. Math. Surveys, 27 (5), 141-192.

SINAI, Ya. G., (1972), Gibbs Measures in Ergodic Theory, Russian Math. Surveys, 27 (4), 21-33.

STOOP, R., and PARISI, J., (1991), Calculation of Lyapunov Exponents Avoiding Spurious Elements, Physica D, 50, pp. 89-94.

STUART, J., and TABOR, M., (1990), The Lagrangian Picture of Fluid Motion, Phil. Trans. R. Soc. Lond. A, 333, pp. 263-271.

ŞUHUBİ, E., (1968), Akışkanlar Mekaniği, Lecture Notes, İstanbul Teknik Üniversitesi, İnşaat Fakültesi Matbaası, 1968.

THOMPSON, J. M. T., and STEWART, H. B., (1986), Nonlinear Dynamics and Chaos, Geometrical Methods for Engineers and Scientist, John Wiley and Sons, Chapter 1.

ÜNAL, G., and GÜNGÖR, F., (1991), İki Çevride Kaotik Karıştırma, VII. Ulusal Mekanik Kongresi, pp. 655-666.

WALKER, G. H., and FORD, J., (1969), Amplitude Instability and Ergodic Behavior for Conservative Nonlinear Oscillator Systems, Phys. Rev., 188 (1), 416-432.

WEISS, J. B., and MCWILLIAMS, J. C., (1991), Nonergodicity of Point Vortices, Phys. Fluids A, 5, pp. 835-844.

WIGGINS, S., (1990), Introduction to Applied Nonlinear Dynamical Systems and Chaos, Springer - Verlag.

WOLF, A., SWIFT, J. B., SWINNEY, H. L., and VASTANO, J. A., (1985), Determining Lyapunov Exponents from a Time Series, Physica D, 16, pp. 285-317.

APPENDIX

Program Listings:

Program listings given below are compiled using Borland International's Turbo C++ version 1. Programs are in a modular form. In order to compile them project function of Turbo C++ must be used or modules must be compiled separatly and then link to make the executable file.

All C++ commands are written with Times font. All comments appears small. Parts of programs appears in between /* and */ are optional and not compiled.

TRA.CPP

This programs displays trajectories. Equations are supplied from an other program (e.g. VORTEX.CPP, SVORTEX.CPP etc.) with the header file equation.h. INTEG.CPP and SETS.CPP must be included.

```
#include "sets.h"
#include "integ.h"
#include "equation.h"
#include "iostream.h"
#include "graphics.h"
#include "stdio.h"
#include "math.h"
#include "conio.h"

int kmax=0,kount=0;        // These variables
double dxsav=0;            // are used in INTEG.CPP
vector K(3);
int N;

void initgraf()            // Initialize graphic adapter
```

```
{
  int a,b,cc;
  detectgraph(&a,&b);
  if (a<0) _error ( "No graphics hardware detected !");
  if (b == EGAHI) b=EGALO;
  initgraph(&a,&b,"");
  cc=graphresult();
  if (cc<0) _error ("initgraph error ");
}

void dispnu(int x, int y, double xini)        // Display number to graphics screen
{
  setviewport (x,y,x+100,y+12,1);
  clearviewport ();
  setviewport (0,0, getmaxx(), getmaxy(),1);
  long l;
  char str[25];
  sprintf(str, "%f", xini);
  outtextxy(x,y,str);
}

void scalebox(int left,                       // Puts scale to screen
    int top,
    int right,
    int bottom,
    float scx,
    float scy,
    int kx=0,
    int ky=0,
    float sx=0.0,
    float sy=0.0)
{
  int min_ch_wi=60;
  float intx, inty;
  char str[30];

  rectangle (left, top, right, bottom);

  // DETERMINING INTERVALS


  for (int f=9; f>=1; f--) {
    for (int n=1; n<=9; n++) {
      intx = n*100000/pow10(f);
```

```
      if (intx > (float)(min_ch_wi/scx) && intx <=((right-left)/2-kx)/scx) goto a;
    }
  }
a:for (f=9; f>=1; f--) {
    for (int n=1; n<=9; n++) {
      inty = n*100000/pow10(f);
      if (inty >=min_ch_wi/scy && inty <= ((bottom-top)/2-ky)/scy) goto b;
    }
  }

b:for (f=0; f<=(int)(((right-left)/2-kx)/scx/intx); f++) {
    line ((int)(left+(right-left)/2-kx+intx*scx*f), bottom,
         (int)(left+(right-left)/2-kx+intx*scx*f), bottom+10);
    line ((int)(left+(right-left)/2-kx+intx*scx*f), top,
         (int)(left+(right-left)/2-kx+intx*scx*f), top-10);
    sprintf(str, "%.4f", intx*f+sx);
     outtextxy((int)(left+(right-left)/2-kx+intx*scx*f-textwidth(str)/2),
bottom+20, str);
    outtextxy((int)(left+(right-left)/2-kx+intx*scx*f-textwidth(str)/2), top-20, str);

  }
  for (f=1; f<=(int)(((right-left)/2+kx)/scx/intx); f++) {
    line ((int)(left+(right-left)/2+kx-intx*scx*f), bottom,
         (int)(left+(right-left)/2+kx-intx*scx*f), bottom+10);
    line ((int)(left+(right-left)/2+kx-intx*scx*f), top,
         (int)(left+(right-left)/2+kx-intx*scx*f), top-10);
    sprintf(str, "%.4f", -intx*f+sx);
     outtextxy((int)(left+(right-left)/2+kx-intx*scx*f-textwidth(str)/2),
bottom+20, str);
    outtextxy((int)(left+(right-left)/2+kx-intx*scx*f-textwidth(str)/2), top-20, str);
  }
  settextstyle(0,1,0);
  for (f=0; f<=(int)(((bottom-top)/2-ky)/scy/inty); f++) {
    line (left, (int)(top+(bottom-top)/2-ky-inty*scy*f), left-6,
         (int)(top+(bottom-top)/2-ky-inty*scy*f));
    line (right, (int)(top+(bottom-top)/2-ky-inty*scy*f), right+6,
         (int)(top+(bottom-top)/2-ky-inty*scy*f));
    sprintf(str, "%.4f", inty*f+sy);
    outtextxy(left-6,(int)(top+(bottom-top)/2-ky-inty*scy*f-textwidth(str)/2),str);

outtextxy(right+16,(int)(top+(bottom-top)/2-ky-inty*scy*f-textwidth(str)/2),str);
  }

  for (f=1; f<=(int)(((bottom-top)/2+ky)/scy/inty); f++) {
```

```
    line (left, (int)(top + (bottom-top)/2 + ky + inty*scy*f), left-6,
          (int)(top + (bottom-top)/2 + ky + inty*scy*f));
    line (right, (int)(top + (bottom-top)/2 + ky + inty*scy*f), right + 6,
          (int)(top + (bottom-top)/2 + ky + inty*scy*f));
    sprintf(str, "%.4f", -inty*f + sy);

outtextxy(left-6,(int)(top + (bottom-top)/2 + ky + inty*scy*f-textwidth(str)/2),str);

outtextxy(right + 16,(int)(top + (bottom-top)/2 + ky + inty*scy*f-textwidth(str)/2
),str);
  }
  settextstyle(0,0,0);
}



main()                          // Program starts from here
{
  FILE *fp;
  double eps, h, hmin, sc, htry, maxx, maxy, xini = 0.0;
  int *c;
  char filename[80];

  cout << "N = ";       cin >> N;
  K.reset(N);
  vector y(2*N), x_no(2), res(2*N), sum(N), t(N);
  Vortex eq(2*N);
  matrix y_no(2,2);

  cout << "Initial Values";
  y.getvalues();
  cout << "vortex strenghts";
  K.getvalues();

  c = new int[N];
  cout << "eps = ";              cin >> eps;
  cout << "h = ";                cin >> h;
  cout << "hmin = ";             cin >> hmin;
  cout << "scale = ";            cin >> sc;
  cout << "htry = ";             cin >> htry;
  cout << "File name: ";         cin >> filename;

  for (int f = 1; f <= N; f++) {
   cout << "c[" << f << "]: ";
```

```
    cin >> c[f];
  }

  initgraf();
  maxx=(int)(getmaxx()/2);
  maxy=(int)(getmaxy()/2);
  scalebox(40, 40, maxx*2-40, maxy*2-40, sc, sc);

  sum.SetAll(0.0);
  do {
    int nok, nbad;

    odeint(y, 2*N, xini, xini+h, eps, htry, hmin, nok, nbad, eq, y_no, x_no);
    xini+=h;
    eq.derivs(xini, y, res);
    for (register int j=1; j<=N; j++) {          //
      sum[j] += K[j]*y[j+N]*res[j]*h;            // This part
    }                                            // calculates
    t = (1/xini)*sum;                            // temparatures
    for (j=1; j<=N; j++)                         //
      dispnu(50, j*11+50, t[j]);                 //

    dispnu(50,50,hamilton(y));
/*  if ((fp=fopen(filename,"a"))==NULL)
      _error ("Cannot open out");
    for (j=1; j<=N; j++)
      fprintf(fp,"%f\n", t[j]);
    fclose(fp);*/
    for (f=1; f<=N; f++) {
      if (c[f]) putpixel((int)(y[f]*sc)+maxx, maxy-(int)(y[f+N]*sc),1);
    }

  } while (!kbhit());
  getch();
  getch();
  closegraph();
  delete c;
  return 0;
}
```

POINCARE SECTIONS PROGRAM

This program calculates and displays the poincare map. Equations are supplied from an other program (e.g. VORTEX.CPP, SVORTEX.CPP etc.) with the header file equation.h. INTEG.CPP and SETS.CPP must be included.

```
#include "sets.h"
#include "integ.h"
#include "equation.h"
#include "math.h"
#include "iostream.h"
#include "stdio.h"
#include "graphics.h"
#include "conio.h"


double dxsav;
int NUM, kount, kmax=0;


inline void dispnu(double xini, int x, int y)
{
  setviewport (x,y,x+100,y+16,1);
  clearviewport ();
  setviewport (0,0, getmaxx(), getmaxy(),1);
  long l;
  char str[25];
  l=(long)(xini*1000);
  ltoa(l,str,10);
  outtextxy(x,y+10,str);
}

void tstep(vector &res, double &xini, hyperplane &p, double eps, double h1,
    double hmin, int a)
{
  Vortex eq(a);
  double check, dum;
  vector v(a);
  matrix y_nouse(2,2);
  vector x_nouse(2);
  int cont=0;
  check=p.ison(res);        // Where is yini relative to p
  for (;;) {
    int nok, nbad;
    dum=check;
    v=res;                  // store yini
    odeint(res, a, xini, xini+h1, eps, h1/2, hmin, nok, nbad, eq, y_nouse,
x_nouse);                   //RK
    check=p.ison(res);
```

```
    dispnu(check,0,0);
    xini+=h1;
    if (check*dum<0) {
      if (cont == NUM) break;
      res=v;
      xini-=h1;
      h1=h1/10;
      cont++;
      check=dum;
    }
  }
}


void initgraf()
{
    int a,b,cc;
    detectgraph(&a,&b);
    if (a<0)
    {
    cout << "No graphics hardware detected !\n";
    exit(1);
  }
  if (b == EGAHI) b=EGALO;
  initgraph(&a,&b,"");
  cc=graphresult();
  if (cc<0)
  {
    cout << "initgraph error :" << grapherrormsg(cc) << "\n";
    exit(1);
  }
}


main()
{
  vector y(8);
  cout << "Initial Values\n";
  y.getvalues();
  vector on(8);
  vector nor(8);
  cout << "A vector on hyperplane?\n";
  on.getvalues();
  cout << "Normal vector of hyperplane?\n";
```

```
  nor.getvalues();
  hyperplane hp(on, nor);
  double eps, h, hmin,sc;

  cout << "eps=";              cin >> eps;
  cout << "h=";                cin >> h;
  cout << "hmin=";             cin >> hmin;
  cout << "scale=";            cin >> sc;
  cout << "Accuracy for tsh"; cin >> NUM;
  FILE *fp;

  initgraf();
  double maxx, maxy;
  maxx = (int)(getmaxx()/2);
  maxy = (int)(getmaxy()/2);

  line(maxx,maxy, maxx, 0);
  line(maxx, maxy, 2*maxx, maxy);
  long t=0;
  double xini=0.0;
  do {
    t++;
    tstep (y, xini, hp, eps, h, hmin, 8);
    if ((fp=fopen("out","a"))==NULL)
      _error ("Cannot open out");
/*    fprintf(fp,"%f %f %f %f %f %f %f %f %f\n", xini, y[1], y[2], y[3], y[4],
            y[5], y[6], y[7], y[8]);
    fclose(fp);  */
    putpixel((int)(y[1]*sc)+maxx, maxy-(int)(y[5]*sc),1);
    dispnu((double)t,0,0);
  } while (!kbhit());
  getch();
  getch();
  closegraph();
  return 0;
}
```

POWER SPECTRA

This program calculates power spectrum of a set of data from an ASCII file. It requires SETS.CPP to be linked.

```
#include <math.h>
#include "sets.h"
#include "graphics.h"
#include "iostream.h"
#include "conio.h"

static float sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)

void memcof(vector &data,int n, int m, float &pm, vector &cof)
{
  int k,j,i;
  float p=0.0;

  vector wk1(n), wk2(n), wkm(m);
  for (j=1;j<=n;j++) p += SQR(data[j]);
  pm=p/n;
  wk1[1]=data[1];
  wk2[n-1]=data[n];
  for (j=2;j<=n-1;j++) {
    wk1[j]=data[j];
    wk2[j-1]=data[j];
  }
  for (k=1;k<=m;k++) {
    float num=0.0,denom=0.0;
    for (j=1;j<=(n-k);j++) {
      num += (wk1[j]*wk2[j]);
      denom += (SQR(wk1[j])+SQR(wk2[j]));
    }
    cof[k]=2.0*num/denom;
    pm *= (1.0-SQR(cof[k]));
    if (k != 1)
      for (i=1;i<=(k-1);i++)
        cof[i]=wkm[i]-cof[k]*wkm[k-i];
    if (k == m) {
      return;
    }
    for (i=1;i<=k;i++) wkm[i]=cof[i];
    for (j=1;j<=(n-k-1);j++) {
      wk1[j] -= (wkm[k]*wk2[j]);
      wk2[j]=wk2[j+1]-wkm[k]*wk1[j+1];
    }
  }
```

```
}

#undef SQR


float evlmem(float fdt, vector &cof, int m, float pm)
{
  int i;
  float sumr=1.0,sumi=0.0;
  double wr=1.0,wi=0.0,wpr,wpi,wtemp,theta;

  theta=6.28318530717959*fdt;
  wpr=cos(theta);
  wpi=sin(theta);
  for (i=1;i<=m;i++) {
    wr=(wtemp=wr)*wpr-wi*wpi;
    wi=wi*wpr+wtemp*wpi;
    sumr -= cof[i]*wr;
    sumi -= cof[i]*wi;
  }
  return pm/(sumr*sumr+sumi*sumi);
}


void initgraf()
{
    int a,b,cc;
    detectgraph(&a,&b);
  if (a<0) _error ( "No graphics hardware detected !");
  if (b == EGAHI) b=EGALO;
  initgraph(&a,&b,"");
  cc=graphresult();
  if (cc<0) _error ("initgraph error ");
}

void dispnu(int x, int y, double xini)
{
    setviewport (x,y,x+100,y+10,1);
    clearviewport ();
    setviewport (0,0, getmaxx(), getmaxy(),1);
    char str[25];
    sprintf(str, "%f", xini);
    outtextxy(x,y,str);
```

```
}


main()
{
 char filename[80];
  int  m, n;
  FILE *fp;
  int sc;
  float lim, add;

  cout << "File name : ";          cin >> filename;
  cout << "n= ";                   cin >> n;
  cout << "m= ";                   cin >> m;
  cout << "sc= ";                  cin >> sc;
  cout << "lim= ";                 cin >> lim;
  cout << "add= ";                 cin >> add;

  if ((fp=fopen(filename,"r"))==NULL) _error("cannot open file");

  vector p(n);
  for (register int j=1; j<=n; j++)
  {
    float dum;
    fscanf(fp, "%f ", &dum);
    p[j]=(double)dum;
  }


  fclose(fp);
  vector cof (m);
  float pm;
  memcof(p, n, m, pm, cof);
  initgraf();
  int maxx=getmaxx()/2;
  int maxy=getmaxy()/2;

  for (float fdt=-lim; fdt<=lim; fdt+=add) {
    float pf;
    pf=evlmem(fdt, cof, m, pm);
    putpixel(maxx+(int)(maxx/lim*fdt), maxy-(int)(pf*sc), 1);
    if (kbhit()) break;
  }
```

```
  getch();
  closegraph();
}
```

LYAPUNOV

This program calculates Lyapunov exponents. It requires SETS.CPP, INTEG.CPP and a equation to be linked.

```
#include "sets.h"
#include "integ.h"
#include "iostream.h"
#include "math.h"
#include "stdlib.h"
#include "conio.h"
#include "stdio.h"
#include "equation.h"


int kmax=0,kount=0;
double dxsav=0;
int N;
vector K(2);


void lyapunov(  double t, int dim, int kmax, double eps,
        vector &x, vector &lam);

main()
{
  cout << "N= ";      cin >> N;
  K.reset(N);
  vector yini(N*2), lam(N*2);
  double t, eps;
  int kmax;

  cout << "Initial conditions";
  yini.getvalues();
  cout << "vortex strenghts";
  K.getvalues();
```

```
  cout << "t=";                cin >> t;
  cout << "Kmax=";             cin >> kmax;
  cout << "eps=";              cin >> eps;

  lyapunov(t, N*2, kmax, eps, yini, lam);

  lam.print();
  return 0;
}

void lyapunov(  double t,          //time span
        int dim,                   //dimension
        int kmax,                  //number of loops
        double eps,
        vector &x,                 //initial condition
        vector &lam)               //return value
{
  matrix u(dim, dim), ny(2,2);
  vector nx(2), dum(dim), y((dim+1)*dim), sum(dim);
  Vortex eq( (dim+1)*dim );
  int nok, nbad;
  register int i,j;
  double tcur = 0.0;
  FILE *fp;

  u.SetAll (0.0);
  for (i=1; i<=dim; i++)
    u(i,i)=1.0;
  sum.SetAll(0.0);

  do {
    for (i=1; i<=dim; i++) {
      y[i]=x[i];
      for (j=1; j<=dim; j++)
        y[i*dim+j]=u(i,j);
    }
    odeint(y, dim*(dim+1), tcur, tcur+t, eps, (double)(t/10), 0.0,
        nok, nbad, eq, ny, nx);
    tcur += t;
    for (i=1; i<=dim; i++) {
      x[i]=y[i];
      for (j=1; j<=dim; j++)
        u(i,j)=y[i*dim+j];
    }
```

```
    dum=u.ortho();
    cout << "dum";
    dum.print();
    for (i=1; i<=dim; i++)
      sum[i] = sum[i]+log(dum[i]);

    lam = (1/tcur)*sum;

    cout << tcur <<"\n";
    lam.print();
    if ((fp=fopen("vor.lya","a"))==NULL) _error("cannot open file");
    for (i=1; i<=2*N; i++)
      fprintf(fp, "%f\n", lam[i]);
    fclose(fp);
  } while (!kbhit());
}
```

SETS.H

This is the header file of SETS.CPP. Every program to be linked with SETS.CPP must include this file with #include "sets.h" statement.

```
#ifndef __SETS_H
#define __SETS_H

#include "iostream.h"
#include "math.h"
#include "stdlib.h"
#include "stdio.h"
#include "string.h"

void _error (char *_str);


/*
                    DEFINITION FOR
                        SET

*/
```

```
class set
{
protected:
  double *_X;
  int _dim;

public:
  set (int _i)                    // constructor allocates memory
  {
    if (_i<1) _error ("Dimension can't be lower than 1 in class set");
    _dim = _i;
    _X = new double[_dim];
    if (!_X) _error ("Memory allocation error in class set constructor");
  }

  set  (const set &_s);           // copy constructor
  ~set () { delete [_dim] _X; }   // destructor

  double &operator[](int _i)
  {
    if (_i<1 || _i>_dim) _error("Out of scope in set::[]");
    return _X[_i - 1];
  }

  void reset(int _i)
  {
    if (_i<1) _error ("Dimension can't be lower than 1 in class set");
       delete [_dim] _X;
    _dim = _i;
    _X = new double[_dim];
    if (!_X) _error ("Memory allocation error in reset");
  }

  int getdim() { return _dim; }
  double *pointer() { return _X-1; }
  void operator=(set &_s);
  int operator==(set &_s);
  int operator!=(set &_s);
  void operator+=(set &_s);
  void operator-=(set &_s);

  void SetAll(double _val);
  void AddAll(double _val);
  void MulAll(double _val);
```

```
  double findmax();
  double findmin();
  void reducedim();
  void getvalues();
  void print();
};



/*

                    DEFINITION FOR
                        VECTOR

*/


class vector : public set
{
public:
  vector(int _i) : set (_i) {}
  vector(const vector &_vec) : set (_vec) {}

  void operator=(vector &_vec);
  vector operator+(vector &_vec);
  vector operator-(vector &_vec);
  double operator*(vector &_vec);
  friend vector operator*(double _v, vector &_vec);
  friend vector operator*(vector &_vec, double _v);

  double norm();
  void unit();

};

/*

                    DEFINITION FOR
                        MATRIX

*/


class matrix:public set
```

```
{
protected:
  int _m,_n;
public:
  matrix(int _i, int _j) : set (_i * _j) { _m=_i; _n=_j;  }
  matrix(const matrix &_mat) : set (_mat) { _m=_mat._m; _n=_mat._n; }

  double &operator()(int _i, int _j)  {
    if (_i<1 || _i>_m || _j<1 || _j>_n)
    _error ("Out of scope in matrix::()");
    return _X[ (_i-1) * _n + _j - 1 ];
  }

  int getm() { return _m; }
  int getn() { return _n; }

  void operator=(matrix &_mat);
  matrix operator+(matrix &_mat);
  matrix operator-(matrix &_mat);
  matrix operator*(matrix &_mat);
  friend matrix operator*(double _v, matrix &_mat);
  friend matrix operator*(matrix &_mat, double _v);

  matrix transpose();
  matrix minor(int _i, int _j);

  vector getcolvec(int _i);
  void setcolvec(int _i, vector &_vec);
  vector getrowvec(int _i);
  void setrowvec(int _i, vector &_vec);

  vector ortho();
  void print();
};

vector operator*(matrix &_mat, vector &_vec);

vector operator*(vector &_vec, matrix &_mat);

void _error(char *_str);

/*
                    DEFINITION FOR
                      HYPERPLANE
```

```
*/

class hyperplane : public set
{
public:
  hyperplane(set &vec) : set(vec) {}
  hyperplane(set &on, set &nor);
  hyperplane(const hyperplane &p) : set(p) {}

  double ison(vector &vec);
  vector normal();
  matrix transfer();
};


#endif
```

SETS.CPP

This is the program where set, vector, matrix and hyperplane objects defined.

```
#include "sets.h"

set::set( const set &_s)          // copy constructor
{
  _dim = _s._dim;
  _X = new double [_dim];
  if (!_X) _error ("Memory allocation error in copy constructor");
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _s._X[_i];
}

void set::operator=(set &_s)
{
  if (_dim != _s._dim) _error ("Incopatible dimensions in =");
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _s._X[_i];
}
```

```
int set::operator==(set &_s)
{
  if (_dim != _s._dim) return -1;
  for (register int _i=0; _i<_dim; _i++)
    if (_X[_i] != _s._X[_i]) return -1;
  return 1;
}

int set::operator!=(set &_s)
{
  if (_dim != _s._dim) return 1;
  for (register int _i=0; _i<_dim; _i++)
    if (_X[_i] != _s._X[_i]) return 1;
  return -1;
}


void set::operator+=(set &_s)
{
  if (_dim != _s._dim) _error ("Incopatible dimensions in +=");
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _X[_i] + _s._X[_i];
}

void set::operator-=(set &_s)
{
  if (_dim != _s._dim) _error ("Incopatible dimensions in -=");
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _X[_i] - _s._X[_i];
}

void set::SetAll(double _v)
{
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _v;
}

void set::AddAll(double _v)
{
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _X[_i] + _v;
}

void set::MulAll(double _v)
```

```
{
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _X[_i] * _v;
}

double set::findmax()
{
  double _temp;
  _temp = _X[0];
  for (register int _i=1; _i<_dim; _i++)
    if (_X[_i] > _temp) _temp = _X[_i];
  return _temp;
}

double set::findmin()
{
  double _temp;
  _temp = _X[0];
  for (register int _i=1; _i<_dim; _i++)
    if (_X[_i] < _temp) _temp = _X[_i];
  return _temp;
}

void set::reducedim()
{
  double *temp;
  temp = new double [_dim-1];
  for (register int i=0; i<_dim-1; i++)
    temp[i]=_X[i+1];
  delete [_dim] _X;
  _X = temp;
  _dim--;
}

void set::getvalues()
{
  cout << "\n";
  double d;
  for (register int _i=0; _i<_dim; _i++) {
    cout << "<" << _i+1 << "> =";
    cin >> d;
    _X[_i]=d;
  }
  cout << "\n";
```

```
}

void set::print()
{
  for (register int _i=0; _i<_dim; _i++)
 ,cout << "<" << _i+1 << "> = " << _X[_i] << "\n";
  cout << "\n";
}

/*

                    DEFINITION FOR
                        VECTOR

*/

void vector::operator=(vector &_vec)
{
  if (_dim != _vec._dim) _error("Incopatible dimensions in vector =");
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _vec._X[_i];
}

vector vector::operator+(vector &_vec)
{
  if (_dim != _vec._dim) _error ("Incopatible dimensions in vector::+");
  vector _temp(_dim);
  for (register int _i=0; _i<_dim; _i++)
    _temp._X[_i] = _X[_i] + _vec._X[_i];
  return _temp;
}

vector vector::operator-(vector &_vec)
{
  if (_dim != _vec._dim) _error ("Incopatible dimensions in vector::-");
  vector _temp(_dim);
  for (register int _i=0; _i<_dim; _i++)
    _temp._X[_i] = _X[_i] - _vec._X[_i];
  return _temp;
}

double vector::operator*(vector &_vec)
{
  if (_dim != _vec._dim) _error ("Incopatible dimensions in vector::*");
```

```
  double _ret=0.0;
  for (register int _i=0; _i<_dim; _i++)
    _ret=_ret+_X[_i]*_vec._X[_i];
  return _ret;
}

vector operator*(double _v, vector &_vec)
{
  vector _temp(_vec._dim);
  for (register int _i=0; _i<_vec._dim; _i++)
    _temp._X[_i] = _vec._X[_i] * _v;
  return _temp;
}

vector operator*(vector &_vec, double _v)
{
  vector _temp(_vec._dim);
  for (register int _i=0; _i<_vec._dim; _i++)
    _temp._X[_i] = _vec._X[_i] * _v;
  return _temp;
}

double vector::norm()
{
  double _ret = 0.0;
  for (register int _i=0; _i<_dim; _i++)
    _ret = _ret+_X[_i]*_X[_i];
  return sqrt(_ret);
}

void vector::unit()
{
  double _d=norm();
  for (register int _i=0; _i<_dim; _i++)
    _X[_i] = _X[_i]/_d;
}

/*
                    DEFINITION FOR
                        MATRIX
*/

void matrix::operator=(matrix &_mat)
```

```
{
  if (_m!=_mat._m || _n!=_mat._n) _error ("Incopatible dimensions in Matrix
=");
  for (register int _i=0; _i<_m*_n; _i++)
    _X[_i] = _mat._X[_i];
},

matrix matrix::operator+(matrix &_mat)
{
  if (_m!=_mat._m || _n!=_mat._n) _error ("Incopatible dimensions in Matrix
+");
  matrix _temp(_m, _n);
  for (register int _i=0; _i<_m*_n; _i++)
    _temp._X[_i] = _X[_i] + _mat._X[_i];
  return _temp;
}

matrix matrix::operator-(matrix &_mat)
{
  if (_m!=_mat._m || _n!=_mat._n) _error ("Incopatible dimensions in Matrix
+");
  matrix _temp(_m, _n);
  for (register int _i=0; _i<_m*_n; _i++)
    _temp._X[_i] = _X[_i] - _mat._X[_i];
  return _temp;
}

matrix matrix::operator*(matrix &_mat)
{
  if (_n != _mat._m) _error ("Incopatable dimensions in matrix*");
  matrix _temp(_m, _mat._n);
  for (register int _i=0; _i<_temp._m; _i++) {
    for (register int _j=0; _j<_temp._n; _j++) {
      double _t=0.0;
      for (register int _k=0; _k<_n; _k++)
        _t = _t + _X[_i+_k*_m] * _mat._X[_k+_j*_mat._m];
      _temp._X[_i+_temp._m*_j] = _t;
    }
  }
  return _temp;
}

matrix operator*(double _v, matrix &_mat)
{
```

```
  matrix _temp(_mat._m, _mat._n);
  for (register int _i=0; _i< _temp._m * _temp._n; _i++)
    _temp._X[_i] = _v * _mat._X[_i];
  return _temp;
}

matrix operator*(matrix &_mat, double _v)
{
  matrix _temp(_mat._m, _mat._n);
  for (register int _i=0; _i< _temp._m * _temp._n; _i++)
    _temp._X[_i] = _v * _mat._X[_i];
  return _temp;
}

matrix matrix::transpose()
{
  matrix _temp(_n, _m);
  for (register int _i=0; _i<_m; _i++) {
    for (register int _j=0; _j<_n; _j++)
      _temp._X[_j*_m+_i] = _X[_j+_n*_i];
  }
  return _temp;
}

matrix matrix::minor(int _i, int _j)
{
  if (_i<1 || _i>_m || _j<1 || _j>_n) _error("Error in matrix::minor");
  _i--;
  _j--;
  matrix _temp(_m-1, _n-1);
  for (register int _il=0; _il<_i; _il++) {
    for (register int _jl=0; _jl<_j; _jl++)
      _temp._X[_il*_temp._n+_jl] = _X[_il*_n+_jl];
    for (_jl=_j; _jl<_temp._n; _jl++)
      _temp._X[_il*_temp._n+_jl] = _X[_il*_n+_j+1];
  }
  for (_il=_i; _il<_temp._m; _il++) {
    for (register int _jl=0; _jl<_j; _jl++)
      _temp._X[_il*_temp._n+_jl] = _X[(_il+1)*_n+_jl];
    for (_jl=_j; _jl<_temp._n; _jl++)
      _temp._X[_il*_temp._n+_jl] = _X[(_il+1)*_n+_j+1];
  }
  return _temp;
}
```

```
vector matrix::getcolvec(int _i)
{
  if (_i<1 || _i>_n) _error ("Error in matrix::getcolvec");
  vector _ret(_m);
  for (register int _il=1; _il<=_m; _il++)
    _ret[_il] = _X[(_il-1)*_n + _i-1];
  return _ret;
}

void matrix::setcolvec(int _i, vector &_vec)
{
  if (_vec.getdim() != _m) _error ("Error a in matrix::setcolvec");
  if (_i<1 || _i>_n) _error ("Error b in matrix::setcolvec");
  for (register int _il=1; _il<=_m; _il++)
    _X[(_il-1)*_n + _i-1] = _vec[_il];
}

vector matrix::getrowvec(int _i)
{
  if (_i<1 || _i>_m) _error ("Error in matrix::getrowvec");
  vector _ret(_n);
  for (register int _il=1; _il<=_n; _il++)
    _ret[_il] = _X[(_i-1)*_n + _il-1];
  return _ret;
}

void matrix::setrowvec(int _i, vector &_vec)
{
  if (_vec.getdim() != _n) _error ("Error a in matrix::setrowvec");
  if (_i<1 || _i>_m) _error ("Error b in matrix::setrowvec");
  for (register int _il=1; _il<=_n; _il++)
    _X[(_i-1)*_n + _il-1] = _vec[_il];
}

vector matrix::ortho ()
{
  double _d;
  vector _ret(_n);
  for (int _k=1; _k<=_n; _k++) {
    _d = (double)0;
    for (register int _i=1; _i<=_m; _i++)
      _d = _d + _X[(_i-1)*_n +_k-1] * _X[(_i-1)*_n +_k-1];
    _d = sqrt(_d);
    _ret[_k]=_d;
```

```
    for (_i =1; _i<=_m; _i++)
      _X[(_i-1)*_n+_k-1] = _X[(_i-1)*_n+_k-1]/ _d;
    for (register int _j=_k+1; _j<=_n; _j++) {
      _d = (double)0;
       for (_i=1; _i<=_m; _i++)
         _d = _d + _X[(_i-1)*_n+_k-1]*_X[(_i-1)*_n+_j-1];
       for (_i=1; _i<=_m; _i++)
         _X[(_i-1)*_n+_j-1]=_X[(_i-1)*_n+_j-1] - _X[(_i-1)*_n+_k-1]*_d;
    }
  }
  return _ret;


void matrix::print()
{
  for (register int _i=0; _i<_m; _i++) {
    for (register int _j=0; _j<_n; _j++)
      cout << " <" << _i+1 << "," << _j+1 << ">: " << _X[_i*_n+_j];
    cout << "\n";
  }
  cout << "\n";
}

vector operator*(matrix &_mat, vector &_vec)
{
  if (_vec.getdim() != _mat.getn()) _error ("Error in vector *(matrix, vector)");
  vector _temp(_mat.getm());
  for (register int _i=1; _i<=_temp.getdim(); _i++) {
    double _d=(double)0;
    for (register int _j=1; _j<=_vec.getdim(); _j++)
      _d=_d+_mat(_i,_j)*_vec[_j];
    _temp[_i]=_d;
  }
  return _temp;
}

vector operator*(vector &_vec, matrix &_mat)
{
  if (_vec.getdim()!=_mat.getm()) _error ("Error in vector *(vector, matrix)");
  vector _temp(_mat.getn());
  for (register int _i=1; _i<=_temp.getdim(); _i++) {
    double _d=(double)0;
    for (register int _j=1; _j<=_vec.getdim(); _j++)
      _d=_d+_mat(_j, _i)*_vec[_j];
```

```
    _temp[_i] = _d;
  }
  return _temp;
}

void _error(char *_str)
{
    cout << _str << "\n\n";
    exit(1);
}

/*
                        DEFINITION FOR
                          HYPERPLANE
*/

hyperplane::hyperplane(set &on, set &nor): set(nor.getdim() + 1)
{
  for (register int i = 0; i < _dim-1; i++)
    _X[i] = nor[i + 1];

  double temp = 0.0;
  for (i = 1; i < _dim; i++)
    temp += on[i]*nor[i];
  _X[_dim-1] = -temp;
}


double hyperplane::ison(vector &vec)
{
  double ret = 0.0;
  for (register int i = 0; i < _dim-1; i++)
    ret += vec[i + 1] * _X[i];
  ret += _X[_dim];
  return ret;
}

vector hyperplane::normal()
{
  vector ret(_dim-1);
  for (register int i = 0; i < _dim-1; i++)
    ret[i + 1] = _X[i];
  return ret;
}
```

```
matrix hyperplane::transfer()
{
  matrix ret(_dim-1, _dim-1);
  vector dum(_dim-1);
  dum=normal();
  ret.setcolvec(1, dum);
  for (register int i=2; i<_dim; i++) {
    for (register int j=1; j<_dim; j++)
      ret(j,i)=(double)(random((int)100)+1);
    }
  ret.ortho();
  dum=normal();
  dum=dum*ret;
  for (i=1; i<_dim; i++)
    _X[i-1] = dum[i];
  return ret;
}
```

INTEG.H

This is the header file of INTEG.CPP. Every program to be linked with INTEG.CPP must include this file with #include "integ.h" statement.

```
#ifndef __INTEG_H
#define __INTEG_H

#include "sets.h"
#include "math.h"


class ODE
{
protected:
  int n;                                  // dimension
public:
  ODE (int a) { n=a; }
  getdim () { return n; }
  virtual void derivs(double, vector &, vector &) {}
};


extern int kount, kmax;
```

```
extern double dxsav;

void odeint(vector &ystart, int nvar, double x1, double x2, double eps,
     double h1,double hmin, int &nok, int &nbad, ODE &eq,
     matrix &yp, vector &xp);
void bsstep(vector &y, vector &dydx, int nv, double &xx, double htry,
     double eps, vector &yscal, double &hdid, double &hnext, ODE &eq);
void mmid(vector &y, vector &dydx, int nvar, double xs, double htot,
     int nstep, vector &yout, ODE &eq);
void rzextr(int iest, double xest, vector &yest, vector &yz, vector &dy,
     int nv, int nuse, matrix &d, vector &x);
void pzextr(int iest, double xest, vector &yest, vector &yz, vector &dy,
     int nv, int nuse, matrix &d, vector &x);
void rk4(vector &y,vector &dydx,int n, double x, double h, vector &yout,
     ODE &eq);
void rkqc(vector &y,vector &dydx, int n, double &x, double htry, double eps,
     vector &yscal, double &hdid, double &hnext,ODE &eq);


#endif
```

INTEG.CPP

Numeric integrators are in this program.

```
#include "integ.h"

void odeint(vector &ystart, int nvar, double x1, double x2, double eps,
     double h1,double hmin, int &nok, int &nbad, ODE &eq, matrix &yp, vector
&xp)
{
  const int MAXSTP=10000;
  const double TINY=1.0e-30;

  register int nstp,i;
  double xsav,x,hnext,hdid,h;
  vector yscal(nvar), y(nvar), dydx(nvar);

  x=x1;
  h=((x2 > x1) ? fabs(h1) : -fabs(h1));
  nok = nbad = kount = 0;
  y=ystart;
```

```
  if (kmax > 0) xsav=x-dxsav*2.0;
  for (nstp=1;nstp<=MAXSTP;nstp++) {
    eq.derivs(x,y,dydx);
    for (i=1;i<=nvar;i++)
      yscal[i]=fabs(y[i])+fabs(dydx[i]*h)+TINY;
    if (kmax > 0) {
      if (fabs(x-xsav) > fabs(dxsav)) {
        if (kount < kmax-1) {
          xp[++kount]=x;
          yp.setcolvec(kount,y);
          xsav=x;
        }
      }
    }
    if ((x+h-x2)*(x+h-x1) > 0.0) h=x2-x;
    bsstep(y,dydx,nvar,x,h,eps,yscal,hdid,hnext,eq);
    if (hdid == h) ++nok; else ++nbad;
    if ((x-x2)*(x2-x1) >= 0.0) {
      ystart=y;
      if (kmax) {
        xp[++kount]=x;
        yp.setcolvec(kount,y);
      }
      return;
    }
    if (fabs(hnext) <= hmin) _error("Step size too small in ODEINT");
    h=hnext;
  }
  _error("Too many steps in routine ODEINT");
}


void bsstep(vector &y, vector &dydx, int nv, double &xx, double htry,
     double eps, vector &yscal, double &hdid, double &hnext, ODE &eq)
{
  const int IMAX=11, NUSE=7;
  const double SHRINK=0.95, GROW=1.2;
  register int i,j;
  double xsav,xest,h,errmax,temp;
  static int nseq[IMAX+1]={0,2,4,6,8,12,16,24,32,48,64,96};

  vector ysav(nv), dysav(nv), yseq(nv), yerr(nv), x(IMAX);
  matrix d(nv, NUSE);
  h=htry;
```

```
  xsav=xx;
  ysav=y;
  dysav=dydx;
  for (;;) {
    for (i=1;i<=IMAX;i++) {
      mmid(ysav,dysav,nv,xsav,h,nseq[i],yseq,eq);
      xest=(temp=h/nseq[i],temp*temp);
      rzextr(i,xest,yseq,y,yerr,nv,NUSE,d,x);
      errmax=0.0;
      for (j=1;j<=nv;j++)
        if (errmax < fabs(yerr[j]/yscal[j]))
          errmax=fabs(yerr[j]/yscal[j]);
      errmax /= eps;
      if (errmax < 1.0) {
        xx += h;
        hdid=h;
        hnext = i==NUSE? h*SHRINK : i==NUSE-1?
          h*GROW : (h*nseq[NUSE-1])/nseq[i];
        return;
      }
    }
    h *= 0.25;
    for (i=1;i<=(IMAX-NUSE)/2;i++) h /= 2.0;
    if ((xx+h) == (xx)) _error("Step size underflow in BSSTEP");
  }
}


void mmid(vector &y, vector &dydx, int nvar, double xs, double htot,
    int nstep, vector &yout, ODE &eq)
{
  register int n;
  double x,h2,h;

  vector ym(nvar), yn(nvar),swap(nvar);
  h=htot/nstep;
  ym=y;
  yn=y+h*dydx;
  x=xs+h;
  eq.derivs(x,yn,yout);
  h2=2.0*h;
  for (n=2;n<=nstep;n++) {
    swap=ym+h2*yout;
    ym=yn;
```

```
    yn=swap;
    x += h;
    eq.derivs(x,yn,yout);
  }
  yout=0.5*(ym+yn+h*yout);
}


void rzextr(int iest, double xest, vector &yest, vector &yz, vector &dy,
      int nv, int nuse, matrix &d, vector &x)
{
  int m1,k,j;
  double yy,v,ddy,c,b1,b;

  vector fx(nuse);
  x[iest]=xest;
  if (iest == 1) {
    yz=yest;
    dy=yest;
    d.setcolvec(1,yest);
  }
  else {
    m1=(iest < nuse ? iest : nuse);
    for (k=1;k<=m1-1;k++)
      fx[k+1]=x[iest-k]/xest;
    for (j=1;j<=nv;j++) {
      yy=yest[j];
      v=d(j,1);
      c=yy;
      d(j,1)=yy;
      for (k=2;k<=m1;k++) {
        b1=fx[k]*v;
        b=b1-c;
        if (b) {
          b=(c-v)/b;
          ddy=c*b;
          c=b1*b;
        } else
          ddy=v;
        if (k != m1) v=d(j,k);
        d(j,k)=ddy;
        yy += ddy;
      }
      dy[j]=ddy;
```

```
      yz[j]=yy;
    }
  }
}


void pzextr(int iest, double xest, vector &yest, vector &yz, vector &dy,
      int nv, int nuse, matrix &d, vector &x)
{
  register int k1, j;
  int m1;
  double q,f2,f1,delta;

  vector c(nv);
  x[iest]=xest;
  dy=yest;
  yz=yest;
  if (iest == 1) {
    d.setcolvec(1,yest);
  } else {
    m1=(iest < nuse ? iest : nuse);
    c=yest;
    for (k1=1;k1<=m1-1;k1++) {
      delta=1.0/(x[iest-k1]-xest);
      f1=xest*delta;
      f2=x[iest-k1]*delta;
      for (j=1;j<=nv;j++) {
        q=d(j,k1);
        d(j,k1)=dy[j];
        delta=c[j]-q;
        dy[j]=f1*delta;
        c[j]=f2*delta;
        yz[j] += dy[j];
      }
    }
    d.setcolvec(m1,dy);
  }
}


void rk4(vector &y,vector &dydx,int n, double x, double h, vector &yout, ODE &eq)
{
  double xh,hh,h6;
```

```
  vector dym(n), dyt(n), yt(n);
  hh=h*0.5;
  h6=h/6.0;
  xh=x+hh;
  yt=y+hh*dydx;
  eq.derivs(xh,yt,dyt);
  yt=y+hh*dyt;
  eq.derivs(xh,yt,dym);
  yt=y+h*dym;
  dym=dym+dyt;
  eq.derivs(x+h,yt,dyt);
  yout=y+h6*(dydx+dyt+2.0*dym);
}


void rkqc(vector &y,vector &dydx, int n, double &x, double htry, double eps,
      vector &yscal, double &hdid, double &hnext,ODE &eq)
{
  const  double  PGROW=-0.20,  PSHRNK=-0.25,  FCOR=0.06666666,
SAFETY=0.9,
                ERRCON=6.0e-4;

  register int i;
  double xsav,hh,h,temp,errmax;

  vector dysav(n), ysav(n), ytemp(n);
  xsav=x;
  ysav=y;
  dysav=dydx;
  h=htry;
  for (;;) {
    hh=0.5*h;
    rk4(ysav,dysav,n,xsav,hh,ytemp,eq);
    x=xsav+hh;
    eq.derivs(x,ytemp,dydx);
    rk4(ytemp,dydx,n,x,hh,y,eq);
    x=xsav+h;
    if (x == xsav) _error("Step size too small in routine RKQC");
    rk4(ysav,dysav,n,xsav,h,ytemp,eq);
    errmax=0.0;
    ytemp=y-ytemp;
    for (i=1;i<=n;i++) {
      temp=fabs(ytemp[i]/yscal[i]);
      if (errmax < temp) errmax=temp;
```

```
    }
    errmax /= eps;
    if (errmax <= 1.0) {
      hdid=h;
      hnext=(errmax > ERRCON ?
        SAFETY*h*exp(PGROW*log(errmax)) : 4.0*h);
      break;
    }
    h=SAFETY*h*exp(PSHRNK*log(errmax));
  }
  y=y+ytemp*FCOR;
}
```

EQUATION.H

Every equation file must include this header file.

```
#ifndef __EQUATION_H
#define __EQUATION_H

#include "sets.h"
#include "math.h"
#include "integ.h"


class Vortex : public ODE
{
public:
  vortex (int a) : ODE (a) {}
  void derivs(double, vector &, vector &);
};

double hamilton (vector &);

#endif
```

VORTEX.CPP

Defines N unbounded point vortex equations.

```
#include "equation.h"
extern vector K;
```

```
extern int N;

void Vortex::derivs(double x, vector &y, vector &dydx)
{
  const double PI=3.141592456;
  double tx, ty;

  for (register int a=1; a<=N; a++) {
    ty=tx=0.0;
    for (register int b=1; b<=N; b++) {
      if (a == b) continue;
      tx+=K[b]*( y[a+N]-y[b+N] ) / ( (y[a]-y[b])*(y[a]-y[b]) +
                (y[a+N]-y[b+N])*(y[a+N]-y[b+N]) );
      ty+=K[b]*( y[a]-y[b] )/ ( (y[a]-y[b])*(y[a]-y[b]) +
                (y[a+N]-y[b+N])*(y[a+N]-y[b+N]) );
    }
    dydx[a]= -tx/2/PI;
    dydx[a+N]= ty/2/PI;
  }
}


double hamilton (vector &y)
{
  double ret=0.0;
  for (register int i=1; i<=N; i++) {
    for (register int j=1; j<=N; j++) {
      if (i == j) continue;
      ret += K[i]*K[j]*log( (y[i]-y[j])*(y[i]-y[j]) +
           (y[i+N]-y[j+N])*(y[i+N]-y[j+N]) );
    }
  }
  return ret;
}
```

VORTEXL.CPP

Jacobian matrix to be used in Lyapunov exponents program.

```
#include "equation.h"
extern vector K;
extern int N;

void Vortex::derivs(double x, vector &y, vector &dydx)
{

  const double PI=3.141592456;
  double tx, ty;

  for (register int i=1; i<=N; i++) {
    ty=tx=0.0;
    for ( register int j=1; j<=N; j++) {
      if (i == j) continue;
      tx+=K[j]*( y[i+N]-y[j+N] ) / ( (y[i]-y[j])*(y[i]-y[j]) +
              (y[i+N]-y[j+N])*(y[i+N]-y[j+N]) );
      ty+=K[j]*( y[i]-y[j] )/ ( (y[i]-y[j])*(y[i]-y[j]) +
              (y[i+N]-y[j+N])*(y[i+N]-y[j+N]) );
    }
    dydx[i]= -tx/2/PI;
    dydx[i+N]= ty/2/PI;
  }
  matrix J(N*2, N*2);       // Jacobian matrix

  for (register int j=1; j<=N; j++) {
    for (i=1; i<=N; i++) {  // dx/dx
      if (i==j) {
        tx = 0.0;
        ty = 0.0;
        for (register int k=1; k<=N; k++) {
          tx += K[k]*(y[j+N]-y[k+N])*(y[j]-y[k])/
          ((y[j+N]-y[k+N])*(y[j+N]-y[k+N])+(y[j]-y[k])*(y[j]-y[k]));

          ty += K[k]*((y[j]-y[k])*(y[j]-y[k])-(y[j+N]-y[k+N])*(y[j+N]-y[k+N]))/
          ((y[j+N]-y[k+N])*(y[j+N]-y[k+N])+(y[j]-y[k])*(y[j]-y[k]));
        }
        J(j,i)=tx/PI;    //  dx/dx
        J(j,i+N)=-ty/2/PI;  //  dx/dy
        J(j+N,i)=-ty/2/PI;  //  dy/dx
        J(j+N,i+N)=-tx/PI;  //  dy/dy
      }
      else {
        J(j,i)=-K[i]*(y[j+N]-y[i+N])*(y[j]+y[i])/
          ((y[j+N]-y[i+N])*(y[j+N]-y[i+N])+(y[j]-y[i])*(y[j]-y[i]))/PI;
```

```
        J(j,i+N) = K[i]*((y[j]-y[i])*(y[j]-y[i])-(y[j+N]-y[i+N])*(y[j+N]-y[i+N]))/
          ((y[j+N]-y[i+N])*(y[j+N]-y[i+N])+(y[j]-y[i])*(y[j]-y[i]))/2/PI;
        J(j+N,i) = J(j,i+N);
        J(j+N,i+N) = -J(j,i);
      }
    }
  }
  for (j=1; j<=2*N; j++) {
    for ( register int i=1; i<=2*N; i++) {
      for (register int k=1; k<=2*N; k++) {
        tx += J(j,k)*y[k*2*N+i];
      }
      dydx[j*2*N+i] = tx;
    }
  }
}
```

CURRICULUM VITAE

Ahmet Kutsi Nircan was born in İzmit in 1967. After completing primary education in İzmit, he has attended Kabataş Erkek Lisesi in İstanbul. In 1984 he has entered İstanbul Technical University, Naval Architecture and Ocean Engineering Faculty, and graduated in 1989.