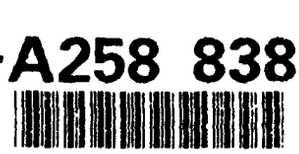

AFIT/GA/ENY/92D-09

EXTENDED LYAPUNOV EXPONENTS FOR SECOND ORDER SYSTEMS

THESIS

Janice M. Horn
Captain, USAF

AFIT/GA/ENY/92D-09

AFIT/GA/ENY/92D-09

EXTENDED LYAPUNOV EXPONENTS FOR SECOND ORDER SYSTEMS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Aerospace Engineering

Janice M. Horn, B.S., Aerospace Engineering
Captain, USAF

December, 1992

Preface

In early 1992, Dr. W. E. Wiesel, a professor at the Air Force Institute of Technology, wrote a paper postulating the existence of an *extended* Lyapunov exponent. He questioned the fact that the classical Lyapunov exponent contained only a real part, while the analogous exponents for the constant coefficient and periodic coefficient cases contain both a real and *imaginary* part. For this thesis, I developed the computer code to calculate the extended Lyapunov exponent, as defined by Wiesel, for second order systems. I then applied this to some classic trial cases. I obtained excellent results for both the constant coefficient and periodic coefficient cases. Thus, the technique presented here validates the existence of the *imaginary* part of the Lyapunov exponent. Since the concept of an extended Lyapunov exponent unifies linear system theory, this work should definitely be continued. In particular, the technique needs to be applied to a wider variety of systems.

In writing the code and analyzing the data, I received a great deal of help from others. I am deeply indebted to my faculty advisor, Dr. W. E. Wiesel, for his untiring patience and assistance. Additionally, I wish to thank Captain Chris Hall for the numerous impromptu discussions about the general subject. A word of thanks also must be sent to Major Robinson, Dr. Phil Beran and Mr. Ron Gordon of the ENY Computational Dynamics and Design Laboratory for the use of the Sun workstations and the tailoring of the software to meet my specific needs.

I must also recognize the endless support that my parents have provided throughout my life. Thank you Mom for instilling in me a love of learning. I wish you could be here to see this, though I know you are here in spirit. Thank you Dad for teaching me the value of stick-to-itiveness. I'm so happy that you can share this with me.

Additionally, I wish to thank my wonderful family. To Fred and Jesse: Thank you, my sons, for your love and understanding during these past months when I couldn't always be there for you. Finally, to my husband, Fred: If it wasn't for your perpetual faith in me and never-ending support, I would not have survived. Thank you, and I love you, Babe!

Janice M. Horn

Table of Contents

List of Figures

List of Tables

AFIT/GA/ENY/92D-09

Abstract

It is shown that the concept of the Lyapunov exponent can be extended to include an *imaginary* part. A numerical technique used to calculate these *extended* Lyapunov exponents for second order systems is presented. There are two requirements to make this extension possible: the definition of a coordinate frame on the tangent space of the differential equation, and an extension of the classical limit, called the *limit of the mean.* An application of the technique to the van der Pol equation for the constant coefficient and periodic coefficient cases is given. The extended Lyapunov exponents found using this technique totally agree with the eigenvalues for the constant coefficient case. In the periodic coefficient case, not only do the extended Lyapunov exponents agree with the Poincaré exponents calculated using standard Floquet theory, they confirm that the imaginary parts of the Poincaré exponents are equal to the quotient $\pm i\frac{(2\pi)}{\tau}$ where τ is the period of the trajectory. This imaginary part is uncertain when using standard Floquet theory. Additionally, fast Fourier transform (FFT) techniques are used to validate the existence of the extended Lyapunov exponent and the values obtained for its imaginary part. These techniques show that the power spectrum of relative motion is discrete for the trial cases presented, with the fundamental frequency almost exactly equal to the calculated imaginary part of the extended Lyapunov exponent. Coupled with the successful comparison of characteristic exponents for the constant coefficient and periodic coefficient cases, this power spectrum serves to decisively validate the existence of the *extended* Lyapunov exponent.

EXTENDED LYAPUNOV EXPONENTS FOR SECOND ORDER SYSTEMS

I. Introduction

Why investigate an *extended* Lyapunov exponent? In virtually every area of the sciences, problems of interest involve system elemental rates of change dependent upon the interaction of the system basic elements. This interaction is universally expressed as a system of first order differential equations

$$\dot{\mathbf{X}}(t) = \mathbf{F}(\mathbf{X}, t) \tag{1.1}$$

This system of equations could model anything from a mechanical system to a chemical reaction or an economic event. Once a particular solution, $\mathbf{X_p}(t)$, is obtained for 1.1, the analyst must then ask the following questions:

1) How stable is this solution ? (i.e. Will some slight variation cause the system to diverge from the particular solution ?)

2) What is the system's fundamental frequency: at what frequency does it possess energy ?

Obviously, the question of stability is paramount in predicting system behavior, but the need to understand a system's driving frequencies is equally important. Improper calculation of a system's frequency components has been the ultimate cause of numerous disasters, such as the collapse of a suspension bridge (8). When a suspension bridge was excited at just the right frequency, the extra energy injected into the system, in addition to the energy it already possesssed at that frequency, caused a total collapse of the structure.

The questions of stability and frequency content are both answered by a system's characteristic exponents, where the real parts of the exponents are the key to the stability of the solution, and the imaginary parts determine the system's fundamental frequency. The eigenvalues are the characteristic exponents for a constant coefficient system, and the Poincaré exponents are the characteristic exponents for a periodic coefficient system. Similarly, the Lyapunov exponents are the characteristic exponents for a general time-varying sytsem; however, classic Lyapunov exponents are defined to be real, not complex numbers. Thus, they yield no frequency information. The *extended* Lyapunov exponents discussed in this thesis contain both real and *imaginary* parts. The following section outlines the classical analysis methods that were used as the basis for the concept of extending the Lyapunov exponent to include an imaginary part.

1.1 Classical Analysis

To analyze the particular solution, one first writes a nearby trajectory as

$$\mathbf{X}(t) = \mathbf{X_p}(t) + \delta\mathbf{X}(t) \tag{1.2}$$

where $\delta\mathbf{X}(t)$ is the first order difference between adjacent trajectories and the particular, or known, trajectory. Substituting equation 1.2 into 1.1 yields

$$\dot{\mathbf{X}}(t) = \dot{\mathbf{X}}_p(t) + \delta\dot{\mathbf{X}}(t) = \mathbf{F}(\mathbf{X_p} + \delta\mathbf{X}, t) \tag{1.3}$$

Expanding $\mathbf{F}$ in a Taylor series about $\mathbf{X_p}$, and assuming $\delta\mathbf{X}(t)$ is small, produces

$$\dot{\mathbf{X}}_p(t) + \delta\dot{\mathbf{X}}(t) \approx \mathbf{F}(\mathbf{X_p}, t) + \left.\frac{\partial\mathbf{F}}{\partial\mathbf{X}}\right|_{\mathbf{X_p}} \delta\mathbf{X} \tag{1.4}$$

Since $\dot{\mathbf{X}}_p(t) = \mathbf{F}(\mathbf{X_p}, t)$, we may cancel it from both sides of 1.4 to obtain the equation of variation

$$\delta\dot{\mathbf{X}}(t) = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{X}}\right|_{\mathbf{X_p}} \delta\mathbf{X}(t) \tag{1.5}$$

The partial derivative of the vector $\mathbf{F}$ with respect to $\mathbf{X}$ is a square matrix generally referred to as a Jacobian matrix, that will hereafter be referred to as $\mathbf{A}(t)$. In essence, $\mathbf{A}(t)$ is the matrix of partial derivatives of the differential equations with respect to the variables of the problem, the elements of $\mathbf{X}$. After evaluation on the particular, or nominal, trajectory $\mathbf{X_p}$, $\mathbf{A}$ is a function of time alone.

Since equation 1.5 has had quadratic terms truncated (due to small $\delta\mathbf{X}$), the character of its solutions governs the *linear stability* of the trajectory $\mathbf{X_p}(t)$. Also, since equation 1.4 is a linear system, its solution can be written as

$$\delta\mathbf{X}(t) = \mathbf{\Phi}(t, t_0)\delta\mathbf{X}(t_0) \tag{1.6}$$

where the fundamental matrix $\mathbf{\Phi}$ obeys the following

$$\dot{\mathbf{\Phi}}(t, t_0) = \mathbf{A}(t)\mathbf{\Phi}(t, t_0) \tag{1.7}$$

$$\mathbf{\Phi}(t_0, t_0) = \mathbf{I} \text{ (the identity matrix)} \tag{1.8}$$

When $\mathbf{X_p}$ is an equilibrium point of 1.1, then $\mathbf{A}$, the associated linear system of 1.1, contains *constant* terms. For linear, constant coefficient systems, the fundamental matrix can be written as

$$\mathbf{\Phi}(t, t_0) = \mathbf{E}\exp(\mathbf{J}(t - t_0))\mathbf{E}^{-1} \tag{1.9}$$

where the eigenvectors of $\mathbf{A}$ form the $\mathbf{E}$ matrix, and the $\mathbf{J}$ matrix, a matrix of constants, is the Jordan normal form of $\mathbf{A}$ with the eigenvalues of $\mathbf{A}$ along its diagonal. The real part of the eigenvalues determine stability: negative real parts imply stabil-

ity, positive real parts indicate instability, and real parts equal to zero imply marginal stability (8). The imaginary parts of the eigenvalues determine the fundamental or oscillation frequencies.

When $\mathbf{X_p}$ is a periodic orbit, $\mathbf{A}$ is a periodic matrix, and Floquet theory is applicable (2). In this instance, the fundamental matrix is written as

$$\mathbf{\Phi}(t, t_0) = \mathbf{E}(t)\exp(\mathbf{J}(t - t_0))\mathbf{E}^{-1}(t_0) \tag{1.10}$$

Now, however, the eigenvector matrix is a function of time, and is periodic. The diagonal elements of $\mathbf{J}$ are now called Poincaré exponents. Due to its periodicity, $\mathbf{E}(t)$ is bounded. Thus, analogous to the eigenvalues in the constant coefficient case, all stability information is found in the Poincaré exponents.

When $\mathbf{X_p}$ is a general trajectory, $\mathbf{A}(t)$ is time varying. For this general case, Lyapunov (6) showed that there are N *real* numbers, Lyapunov exponents, which determine the local exponential rates of growth away from, or convergence to the reference trajectory. Accepted algorithms for calculating Lyapunov exponents have been reported by Benettin, et al (3) and by Shimada and Nagashimi (9). Extending this general case exponent to include an imaginary part is the heart of this thesis.

1.2 Problem Statement

As stated in the previous section, Lyapunov showed the existence of N *real*, not complex, numbers characterizing the growth away from, or convergence to a particular solution. In his paper "Extended Lyapunov Exponents"(11), Wiesel posed the following question: Given the fact that in the constant coefficient and periodic coefficient cases, the characteristic exponents are *complex* numbers, why are the Lyapunov exponents limited to being *real* ?

Thus, in light of 1.9 and 1.10, Wiesel proposed a fundamental matrix decomposition for the general case to be

$$\mathbf{\Phi}(t, t_0) = \mathbf{E}(t)\exp(\mathbf{J}(t - t_0))\mathbf{E}^{-1}(t) \tag{1.11}$$

where the eigenvector matrix $\mathbf{E}(t)$ would take on the character of $\mathbf{A}(t)$: constant, periodic, multiply periodic, or aperiodic, and the diagonal elements of the constant Jordan matrix $\mathbf{J}$ would govern stability.

The main objectives of this study are to calculate, and validate the existence of *extended* Lyapunov exponents (i.e. ones that include an imaginary part), for some trial cases. The existence of an extended Lyapunov exponent would unify the theory of linear systems for all cases. Computer programs need to be developed to apply the technique to some trial cases. During this study, the trial cases will be limited to second-order systems . After development of the computer programs, the next step will be to apply them to a second order system and analyze some classic behaviors: the constant coefficient and periodic coefficient cases. Comparison of the calculated extended Lyapunov exponents to the eigenvalues of the $\mathbf{A}$ matrix for linearization about an equilibrium point will validate the constant coefficient case. Linearizing about the limit cycle, provided a limit cycle exists, will serve to validate the periodic coefficient case. Naturally, for the periodic coefficient case, the comparison will be against the Poincaré exponents which are calculated using Floquet theory.

1.3 Overview of Thesis Content

The following chapter discusses the derivation of the *extended* Lyapunov exponent. It includes both an heuristic derivation and a derivation based on Lyapunov's original definition. Additionally, it covers the development of the limit *of* the mean and the coordinate frame definition. Chapter III contains a brief overview

of the numerical algorithm development and the associated FORTRAN-coded programs/subroutines.

In Chapter IV, the reader will find an application of the technique to a second order system, van der Pol's equation, with linearization about the equilibrium point and the limit cycle. Also, Chapter IV contains the FFT analysis used to validate the imaginary parts of the extended Lyapunov exponents. The last chapter contains the author's conclusions and recommendations.

II. Derivation of the Extended *Lyapunov Exponent*

Wiesel's (11) arguments for extending the Lyapunov exponent will be presented in this chapter. The first section will contain two derivations of the extended Lyapunov exponents, the first heuristic and the second generalizing Lyapunov's original definition. It will be shown in the second section that the definition of a coordinate frame is essential to the calculation of an imaginary part of an exponent, even in the constant coefficient case. An extended definition of a limit, the *limit of the mean*, will be developed in the third section. Finally, the last section imparts Wiesel's original conjecture.

2.1 The Fundamental Matrix and the Extended *Lyapunov Exponent*

A basic property of the fundamental matrix $\mathbf{\Phi}$ is

$$\dot{\mathbf{\Phi}}(t, t_0) = \mathbf{A}(t)\mathbf{\Phi}(t, t_0) \tag{2.1}$$

where t is the current time, and t_0 is the initial time. Equation 2.1 is a general, time dependent linear system with the initial condition, $\mathbf{\Phi}(t_0, t_0) = \mathbf{I}$ (the identity matrix). As will be shown in the forthcoming section, this initial condition is crucial.

Using standard matrix eigenvector decomposition

$$\mathbf{\Phi}(t, t_0) = \mathbf{E}(t)\mathbf{\Lambda}(t)\mathbf{E}^{-1}(t) \tag{2.2}$$

where $\mathbf{E}(t)$ and $\mathbf{\Lambda}(t)$ are the fundamental matrix eigenvector and eigenvalue matrices, respectively. It is assumed that the eigenvalues λ_i on the diagonal of the diagonal matrix $\mathbf{\Lambda}$ are all distinct, so that $\mathbf{\Lambda}$ is not a Jordan form. Since $\mathbf{\Phi}(t_0, t_0) = \mathbf{I}$, $\mathbf{\Lambda}(t_0)$ must equal the identity matrix, and $\mathbf{E}(t_0)$ should be some invertible matrix. If the eigenvectors are normalized, then any long term stability information resides within $\mathbf{\Lambda}$.

To further continue the analogy with 1.11, a diagonal matrix $\mathbf{\Omega}$ is defined by

$$\omega_i(t - t_0) = \ln \lambda_i(t) \tag{2.3}$$

where the initial condition $\mathbf{\Lambda}(t_0) = \mathbf{I}$ is satisfied. Equation 2.2 can now be rewritten as

$$\mathbf{\Phi}(t, t_0) = \mathbf{E}(t)\exp(\mathbf{\Omega}(t)(t - t_0))\mathbf{E}^{-1}(t) \tag{2.4}$$

Comparison of equation 2.4 to 1.11 shows that the exponential rates of change of the solution vectors over the interval (t_0, t) are given by $\omega_i(t)$, where

$$\omega_i(t) = \frac{1}{(t - t_0)} \ln \lambda_i(t) \tag{2.5}$$

Then, should the limit exist as time approaches infinity, the long-term system response is defined by

$$\omega_i(\infty) = \lim_{t \to \infty} \omega_i(t) \tag{2.6}$$

where the real parts of $\omega_i(t)$ are the average exponential rates of increase of the solution, and the imaginary parts are the average oscillation frequencies of the solution with respect to the eigenvector basis. (The eigenvectors are independent, and span the solution vector space.)

Thus, the real parts are the classic Lyapunov exponents of the system. The imaginary parts must exist in comparison to the constant coefficient and periodic coefficient cases, though no mention of them was found in any literature searches conducted. Therefore, Wiesel dubbed $\omega_i(\infty)$ as the *extended* Lyapunov exponents.

The following derivation will use a different tack to arrive at equation 2.6: generalizing Lyapunov's original definition. First, taking equation 2.4, and post multiplying the left and right sides by $\mathbf{E}(t)$ yields

$$\mathbf{\Phi}(t, t_0)\mathbf{E}(t) = \mathbf{E}(t)\exp[\mathbf{\Omega}(t)(t - t_0)] \tag{2.7}$$

or,

$$e_i(t)\exp[\omega_i(t)(t - t_0)] = \mathbf{\Phi}(t, t_0)e_i(\mathbf{t}) \tag{2.8}$$

where $e_i(t)$ are columns of $\mathbf{E}(t)$. The revealing form of equation 2.8, upon comparison to 1.6, shows that each column vector of equation 2.8 is a solution to

$$\dot{u}_i(t) = \mathbf{A}(t)u_i(t) \tag{2.9}$$

for the following initial conditions and final values:

$$u_i(t_0) = e_i(t) \tag{2.10}$$

$$u_i(t) = e_i(t)\exp(\omega_i(t)(t - t_0)) \tag{2.11}$$

It is assumed that $u_i(t_0)$ is normalized.

Next, Lyapunov (6) showed that for $\dot{u}_i(t) = \mathbf{A}(t)u_i(t)$, N *real* Lyapunov exponents exist, which obey

$$\mu_i = \lim_{t\to\infty} \sup \frac{\ln |\mathbf{\Phi}(t, t_0)u_i(t_0)|}{(t - t_0)} \tag{2.12}$$

However, from the definition of the fundamental matrix and equation 2.11, it is known that

$$\mathbf{\Phi}(t, t_0)u_i(t_0) = u_i(t) \tag{2.13}$$

and

$$u_i(t) = e_i(t)\exp(\omega_i(t)(t - t_0)) \tag{2.14}$$

Therefore,

$$\mu_i = \lim_{t\to\infty} \sup \frac{\ln |e_i(t)\exp(\omega_i(t)(t - t_0))|}{(t - t_0)} \tag{2.15}$$

$$= \lim_{t\to\infty} \sup \left[\frac{\ln |e_i(t)|}{(t - t_0)} + \frac{\ln |\exp(\omega_i(t)(t - t_0))|}{(t - t_0)}\right] \tag{2.16}$$

Since the eigenvectors were assumed to be normalized, the following statement is true:

$$\ln |e_i(t)| = 0 \tag{2.17}$$

Thus,

$$\mu_i = \lim_{t\to\infty} \sup \frac{\ln |\exp(\omega_i(t)(t - t_0))|}{(t - t_0)} \tag{2.18}$$

Now, let $\omega_i(t) = \alpha + i\beta$, where $i = \sqrt{-1}$.

$$\begin{aligned} \Longrightarrow \mu_i &= \lim_{t\to\infty} \sup \frac{\ln |\exp[(\alpha + i\beta)(t - t_0)]|}{(t - t_0)} && (2.19)\\ &= \lim_{t\to\infty} \sup \frac{\ln\left[[|\exp(\alpha(t - t_0))|][|\exp(i\beta(t - t_0))|]\right]}{(t - t_0)} && (2.20)\\ &= \lim_{t\to\infty} \sup \left[\frac{\ln |\exp(\alpha(t - t_0))|}{(t - t_0)} + \frac{\ln |\exp(i\beta(t - t_0))|}{(t - t_0)}\right] && (2.21) \end{aligned}$$

Making use of Euler's formula (4): $e^{i\theta} = \cos(\theta) + i\sin(\theta)$,

$$\mu_i = \lim_{t\to\infty} \left[\sup \frac{\ln |\exp(\alpha(t - t_0))|}{(t - t_0)} + \sup \frac{\ln |\cos(\beta(t - t_0)) + i\sin(\beta(t - t_0))|}{(t - t_0)}\right] \tag{2.22}$$

For *suprema* conditions, the second ln term vanishes. Thus,

$$\begin{aligned} \mu_i &= \lim_{t\to\infty} \sup \frac{\ln |\exp(\alpha(t - t_0))|}{(t - t_0)} \\ &= \lim_{t\to\infty} \sup(\alpha) \\ &= \lim_{t\to\infty} \sup Re\ \omega_i(t) \end{aligned} \tag{2.23}$$

Since the eigenvectors are independent and span the solution space (i.e. form a basis), the N trial trajectories span the space of initial conditions, and any other trajectory with $|u(t_0)| = 1$ can be written as a linear combination of $u_i(t_0)$. This implies that the classical Lyapunov exponents are (provided the limit exists)

$$\mu_i = \lim_{t\to\infty} \sup \frac{1}{(t - t_0)} Re\ \ln \lambda_i(t) \tag{2.24}$$

where λ_i are the eigenvalues of $\mathbf{\Phi}$.

It is highly unlikely that $u_i(t)$ and $u_i(t_0)$ will be parallel in the classic case, thus the norm is required. Since the product of a matrix and a vector equals a vector, and the norm of a vector is equal to a real number, it is obvious from equation 2.12 that it is this norm that limits the classical Lyapunov exponents to being real, rather than complex, numbers. To show that this is true, the reader should note the following definitions for the norm of a vector.

First, if the vector $\mathbf{y}$ is a vector of real numbers, its norm is defined (10)

$$|\mathbf{y}| \equiv \sqrt{y_1^2 + y_2^2 + \ldots + y_n^2} \tag{2.25}$$

where y_i are the elements of $\mathbf{y}$, and the norm of $\mathbf{y}$ is a real number. Next, if $\mathbf{y}$ contains complex numbers, the following definition for its norm applies (10)

$$|\mathbf{y}| \equiv \sqrt{y_1\overline{y}_1 + y_2\overline{y}_2 + \ldots + y_n\overline{y}_n} \tag{2.26}$$

where $\overline{y}_i$ is the complex conjugate of y_i. Thus, since the product of a complex number and its conjugate is always a positive real number, the norm of the complex vector $\mathbf{y}$ is real.

Now, since this derivation has used the eigenvectors as trial trajectories, the norm becomes unnecessary. This fact is highlighted by equation 2.8. Which shows that when propogating u_i from t_0 to t, $u_i(t_0) = e_i(t)$ is simply multiplied by a scalar. Therefore, $u_i(t)$ is parallel to $u_i(t_0)$, which eliminates the need for a norm.

Elimination of the norm from 2.12 yields

$$\begin{aligned} \mu_i &= \lim_{t\to\infty} \sup \frac{\ln\left[\exp(\omega_i(t)(t - t_0))\right]}{(t - t_0)} \\ &= \lim_{t\to\infty} \omega_i(t) \\ &= \omega_i(\infty) \end{aligned} \tag{2.27}$$

Equation 2.27 is identical to the heuristically derived 2.6.

2.2 The Need for Coordinates

As the previous section illustrated, the definition of the classical Lyapunov exponents assumes that the tangent space to the differential equation $\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X}, t)$ possesses a norm, but *not* a coordinate frame. This section will use two simple examples to illustrate the crucial role that the choice of a coordinate frame plays in the extended Lyapunov exponent.

First, consider the simple autonomous linear system, where

$$\mathbf{A} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.28}$$

Two solutions to $\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi}$ are

$$\begin{aligned} \mathbf{\Phi}_1(t) &= \begin{pmatrix} e^{-t} & 0 \\ 0 & e^{t} \end{pmatrix} \\ \mathbf{\Phi}_2(t) &= \begin{pmatrix} 0 & -e^{-t} \\ e^{t} & 0 \end{pmatrix} \end{aligned} \tag{2.29}$$

In either solution, one column vector grows with logarithmic rate of $+1$, and the other decreases with logarithmic rate -1. Clearly then, the Lyapunov exponents, which are also the eigenvalues of $\mathbf{A}$, are ± 1 in either case.

The next step is to calculate the extended Lyapunov exponents for the two solutions. First, the calculation of the eigenvalues of $\mathbf{\Phi}$ must be accomplished. To do this, one must solve the characteristic equation which is defined by

$$\det\{\mathbf{\Phi} - \lambda\mathbf{I}\} = \begin{vmatrix} \phi_{11} - \lambda & \phi_{12} \\ \phi_{21} & \phi_{22} - \lambda \end{vmatrix} = 0 \tag{2.30}$$

For Φ_1, this determinant yields

$$\lambda^2 + \lambda(-e^{-t} - e^{t}) + 1 = 0 \tag{2.31}$$

$$\begin{aligned} \Longrightarrow \lambda_i &= \left[\frac{(e^{-t} + e^{t}) \pm \sqrt{(e^{-t} - e^{t})^2}}{2}\right] \\ &= \left[\frac{(e^{-t} + e^{t}) \pm (e^{-t} - e^{t})}{2}\right] \end{aligned} \tag{2.32}$$

Thus, $\lambda_1 = e^{-t}$ and $\lambda_2 = e^{t}$.

Solving for the extended Lyapunov exponents yields

$$\omega_1 = \lim_{t\to\infty} \frac{1}{t} \ln e^{t} = \lim_{t\to\infty} \frac{t}{t} = 1 \tag{2.33}$$

$$\omega_2 = \lim_{t\to\infty} \frac{1}{t} \ln e^{-t} = \lim_{t\to\infty} \frac{-t}{t} = -1 \tag{2.34}$$

(i.e. the expected values).

Next, the characteristic equation for Φ_2 is

$$\lambda^2 + 1 = 0 \tag{2.35}$$

$$\Longrightarrow \lambda_i = \pm i \tag{2.36}$$

Another Euler identity states that the polar form of a complex number $\alpha + i\beta$ is expressed as $re^{i\theta}$ where $r = \mathrm{sqrt}(\alpha^2 + \beta^2)$ and $\theta = \arctan\left\{\frac{\beta}{\alpha}\right\}$.

$$\Longrightarrow \lambda_1 = +i = e^{i\pi} \text{ and } \lambda_2 = -i = e^{-i\pi} \tag{2.37}$$

Using the polar form of λ_i to calculate the extended Lyapunov equations yields the following:

$$\omega_1 = \lim_{t\to\infty} \frac{1}{t} \ln e^{+i\pi} = \lim_{t\to\infty} \frac{+i\pi}{t} \tag{2.38}$$

$$\omega_2 = \lim_{t\to\infty} \frac{1}{t} \ln e^{-i\pi} = \lim_{t\to\infty} \frac{-i\pi}{t} \tag{2.39}$$

Obviously, evaluating equations 2.38 and 2.39 as t approaches infinity does not yield the correct exponents. However, the derivation of the extended Lyapunov exponents stipulated that $\mathbf{\Phi}(t_0, t_0) \equiv \mathbf{I}$. Evaluating $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ at $t = 0$ yields

$$\mathbf{\Phi}_1(t_0, t_0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.40}$$

$$\mathbf{\Phi}_2(t_0, t_0) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{2.41}$$

Thus, defining $\mathbf{\Phi}(t_0, t_0) = \mathbf{I}$ is absolutely necessary to ensure the correct propagation of the initial conditions at $t = 0$ to the coordinate-based final conditions at time t. Therefore, the requirement $\mathbf{\Phi}(t_0, t_0) = \mathbf{I}$ selects a *unique* solution to $\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi}$, and maintains coordinate continuity from $t = 0$ to time t. Additionally, the reader should note that for classical Lyapunov exponent calculation, any solution to $\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi}$ suffices to determine the length change of individual vector solutions.

To further illustrate the necessity of coordinates, consider the linear constant coefficient system represented by

$$\mathbf{A} = \begin{pmatrix} 0 & \omega \\ -\omega & 0 \end{pmatrix} \tag{2.42}$$

Upon inspection, two solutions to $\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi}$ are

$$\mathbf{\Phi}_1(t) = \begin{pmatrix} \cos\omega t & \sin\omega t \\ -\sin\omega t & \cos\omega t \end{pmatrix}$$

$$\mathbf{\Phi}_2(t) = \begin{pmatrix} \cos(\omega t + \psi) & \sin(\omega t + \psi) \\ -\sin(\omega t + \psi) & \cos(\omega t + \psi) \end{pmatrix} \tag{2.43}$$

Both matrices have column solution vectors whose norms are one for all time. Thus, the classic Lyapunov exponents are zero for this system, and the eigenvalues of **A** are $\pm i\omega$ (purely complex). Therefore, it is expected that the extended Lyapunov exponents will be $\pm i\omega$.

Now, to calculate the extended Lyapunov exponents for the system defined by $\mathbf{\Phi}_1$, the first thing to do is to calculate the eigenvalues of $\mathbf{\Phi}_1$:

$$\lambda^2 + \lambda(-2\cos\omega t) + 1 = 0 \tag{2.44}$$

$$\begin{aligned} \Longrightarrow \lambda_i &= \frac{2\cos\omega t \pm \sqrt{4\cos^2\omega t - 4}}{2} \\ &= \cos\omega t \pm \sqrt{\cos^2\omega t - 1} \\ &= \cos\omega t \pm \sqrt{-\sin^2\omega t} \\ &= \cos\omega t \pm i\sin\omega t \end{aligned} \tag{2.45}$$

Using these values the extended Lyapunov Exponents for this system are found to be

$$\begin{aligned} \omega_i &= \lim_{t\to\infty} \frac{1}{t} \ln(\cos\omega t \pm i\sin\omega t) \\ &= \lim_{t\to\infty} \frac{1}{t} \ln e^{\pm i(\omega t)} \\ &= \lim_{t\to\infty} \frac{1}{t} (\pm i\omega t) = \pm i\omega \end{aligned} \tag{2.46}$$

Hence, $\mathbf{\Phi}_1$ produced the expected values: the eigenvalues of the **A** matrix.

Similar treatment of $\mathbf{\Phi}_2$ yields a dependence on ψ:

$$\omega_i = \lim_{t\to\infty} \frac{1}{t}(\pm i(\omega t + \psi)) \tag{2.47}$$

If $\psi = 0$, there is total agreement with the expected values; however, for $\psi = -\omega t$, the extended Lyapunov exponents equal zero.

Thus, again it is necessary to choose a coordinate frame. Evaluating $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ at $t = 0$ produces

$$\mathbf{\Phi}_1(t_0, t_0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.48}$$

$$\mathbf{\Phi}_2(t_0, t_0) = \begin{pmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{pmatrix} \tag{2.49}$$

This implies that in order to ensure correct propagation from $t = 0$ to time t, the requirement that $\mathbf{\Phi}(t_0, t_0) = \mathbf{I}$, which implies $\psi = 0$, must be enforced.

The previous examples show that even equation 1.9, the constant coefficient system solution, is a coordinate-based solution. Additionally, it has been shown that the proposed method for calculating the extended Lyapunov exponents **will** produce the correct system roots, both the real and imaginary parts.

These examples also show that without a coordinate-based description of motion, it is meaningless to ever compute an oscillation frequency, the imaginary part of the appropriate exponent, for even a constant coefficient system. So, there should be no surprise that a coordinate frame must be chosen for the general case.

2.3 The Limit of the Mean

There are times when the extended Lyapunov exponent functions do not necessarily possess limits in the classical sense, as time goes to infinity. This section will introduce an apparently new type of limit called the *limit of the mean* (l.o.m.)

(11). Then, it will be argued that, in the sense defined by the l.o.m., the limit for those extended Lyapunov exponent functions does indeed exist.

Starting out with the definition of the mean value of a function on the interval (t_0, t):

$$\langle f \rangle (t) \equiv \frac{1}{(t - t_0)} \int_{t_0}^{t} f(\tau) d\tau \tag{2.50}$$

Holding the lower limit t_0 constant, the limit of the mean will be defined as

$$\underset{t \to \infty}{\text{l.o.m.}} f(t) \equiv \lim_{t \to \infty} \langle f \rangle (t) = \lim_{t \to \infty} \frac{1}{(t - t_0)} \int_{t_0}^{t} f(\tau) d\tau \tag{2.51}$$

Apostol's *limit in the mean* (l.i.m.) (1), which applies to the convergence of a sequence of functions to a limiting function, and Cesaro's summability for an infinite series, are conceptually similar to the limit *of* the mean. In his paper, Wiesel (11) asserted the following fundamental result and proof:

Theorem: For any integrable f(t), if the limit

$$\lim_{t\to\infty} f(t) = f_\infty \tag{2.52}$$

exists, then the limit of the mean also exists and equals f_∞.

If $\lim_{t\to\infty} f(t)$ exists, then for any ϵ there exists a T such that when $t > T$, $|f(t) - f_\infty| < \epsilon$. Splitting the integral of 2.47 yields

$$\underset{t\to\infty}{\text{l.o.m.}} f(t) = \lim_{t\to\infty} \int_t^T f(\tau)d\tau + \lim_{t\to\infty} \int_T^t f(\tau)d\tau \tag{2.53}$$

Assuming f(t) is integrable, the first integral is finite and constant for a given value T. Thus, its limit is trivially zero. The second integral is bounded by

$$\frac{1}{(t-t_0)}\int_T^t (f_\infty - \epsilon)d\tau < \frac{1}{(t-t_0)}\int_T^t f(\tau)d\tau < \frac{1}{(t-t_0)}\int_T^t (f_\infty + \epsilon)d\tau \tag{2.54}$$

Performing the outer integration generates the following:

$$(f_\infty - \epsilon)\frac{(t-T)}{(t-t_0)} < \frac{1}{(t-t_0)}\int_T^t f(\tau)d\tau < (f_\infty + \epsilon)\frac{(t-T)}{(t-t_0)} \tag{2.55}$$

Evaluating 2.55 as $t \to \infty$ yields

$$(f_\infty - \epsilon) < \lim_{t\to\infty} \frac{1}{(t-t_0)}\int_T^t f(\tau)d\tau < (f_\infty + \epsilon) \tag{2.56}$$

Using the definition of the limit of the mean, this implies that

$$(f_\infty - \epsilon) < \underset{t\to\infty}{\text{l.o.m.}} f(t) < (f_\infty + \epsilon) \tag{2.57}$$

However, since this is true for any ϵ, then

$$\underset{t\to\infty}{\text{l.o.m.}} f(t) = f_\infty \tag{2.58}$$

Q.E.D.

Thus, similar to the concept of the limit *in* the mean, the limit *of* the mean may exist when no classical limit exists. To illustrate this, it is easily seen that

$$\underset{t\to\infty}{\text{l.o.m.}} \sin t = 0 \tag{2.59}$$

even though no traditional limit exists. The limit of the mean extends existence to a broader class of functions, but not an unreasonably broad class. For example, the limit of the mean does *not* exist for $t \sin t$, nor does it exist for any monotonic unbounded function.

2.4 Extended Lyapunov Exponents (Final Form)

In section 2.1, the extended Lyapunov exponent was defined by equation 2.6. If the eigenvalues, $\lambda_i(t)$, of the fundamental matrix grow exponentially in the long term, then $\ln \lambda_i$ grows no faster than linearly with time. Therefore, $\omega_i(t)$ may perhaps be oscillatory, but it will have no long term tendency to grow. Thus, $\omega_i(t)$, in all likelihood, belongs to the class of functions for which the limit of the mean exists.

Hence, Wiesel's (11) final conjecture: The classical Lyapunov exponent is the real part of

$$\begin{aligned} \omega_i(\infty) &= \underset{t\to\infty}{\text{l.o.m.}} \omega_i(t) \\ &= \underset{t\to\infty}{\text{l.o.m.}} \frac{\ln \lambda_i(t)}{(t - t_0)} \\ &= \frac{1}{t}\int_{t_0}^{t} \frac{\ln \lambda_i(\tau)}{\tau} d\tau \end{aligned} \tag{2.60}$$

where the limit of the mean must be calculated over extremal pathways through a succession of eigenvalue bifurcations, and the imaginary part denotes the mean oscillation frequency with respect to the chosen coordinate frame. Obviously, the real part is the *average* exponential rate of increase of the solution. The terms *extremal pathways* and *bifurcations* will be addressed further in future chapters.

Validating this conjecture is the main thrust of this thesis. If true, it could lead to a unification of the theory of linear systems.

III. Numerical Algorithm Development

This chapter describes the algorithm used to compute the extended Lyapunov exponents for second order systems, and the associated FORTRAN-coded programs/subroutines. Generally speaking, a predictor-corrector numerical integration scheme (Haming) is used to perform the following integrations:

$$\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X}, t) \tag{3.1}$$

$$\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi} \tag{3.2}$$

$$\omega_i = \frac{1}{t}\int \frac{\ln \lambda_i(\tau)}{\tau} d\tau \text{ (the extended Lyapunov exponent)} \tag{3.3}$$

The *main* program reads the following inputs: initial conditions, system parameters, the maximum time of integration, and the number of steps, and it sets up the output files. Next, it initializes the fundamental matrix $\mathbf{\Phi}$ to the identity matrix, and the integrals of the eigenvalues to zero. The "length" of the vector to be integrated is then defined in the main program, and the timestep is calculated. Then, the numerical integrator, Haming, is initialized. A fourth order predictor-corrector needs four values of the state vector to perform the integration, and at $t = 0$, there is only one (the initial conditions). Haming uses a Picard iteration to get the other three values. Provided this initialization is successful, the main program begins the full scale integration by calling Haming again. After Haming returns each value, the main

program calls the subroutine that checks for eigenvalue bifurcations (SPLIT) and the subroutine that adjusts the phase of the eigenvalues (PHASE). Finally, the *main* program computes the actual extended Lyapunov exponents. All of the required programs/subroutines used for calculating the extended Lyapunov exponents may be found in Appendix A, and are summarized in Table 3.1.

SUBROUTINE	PURPOSE	CALLED BY
Haming	Numerical integration	Main program
		SPLIT
RHS	Calculates right sides of equations 3.1 - 3.3	Haming
AMAT	Calculates the elements of the **A** matrix	RHS
CALEIG	Calculates complex logs of eigenvalues of Φ	RHS
SPLIT	Checks for bifurcations and if one has occurred, restarts Haming	Main program
SPLIT2	Determines bifurcation time and type	SPLIT
	Extrapolates log eigenvalues into and out of bifurcation	
	Swaps root labels if necessary	
PHASE	Fixes phase on current eigenvalues if necessary	Main program
		SPLIT2
CHECKS	Performs reasonableness checks on eigenvalues and averaged integrals	SPLIT

Table 3.1. Subroutines for Calculating the Extended Lyapunov Exponents

The first step in the algorithm is to set up the equations to be integrated 3.1 to 3.3. The next three sections discuss this *setup* in detail for second order systems. Section 4 discusses the development of the subroutine to calculate the eigenvalues of the Φ matrix, and the last section addresses potential problems.

3.1 Deriving the General Form

Starting with a second-order ordinary differential equation, one must transform it into the form $\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X}, t)$. For example, let the following represent the original differential equation where a, b may be functions of x and/or t.

$$\ddot{x} + a\dot{x} + bx = 0 \tag{3.4}$$

Now, let

$$\begin{aligned} x_1 &= x \\ &\Rightarrow \dot{x}_1 = \dot{x} \end{aligned}$$

and

$$\begin{aligned} x_2 &= \dot{x}_1 \\ &\Rightarrow \dot{x}_2 = \ddot{x} = (-ax_2 - bx_1) \end{aligned}$$

This yields,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{Bmatrix} 0 & 1 \\ -b & -a \end{Bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{F}(\mathbf{X}, t) \tag{3.5}$$

or,

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -bx_1 - ax_2 \end{aligned}$$

3.2 Calculating the Right-Hand Sides (RHS)

The right-hand side of 3.6 is calculated in the subroutine RHS (see Appendix A), which is called by Haming. Additionally, RHS calculates the terms in integral 3.2. The calculation of the **A** matrix is discussed in the following section, and is performed in the subroutine AMAT; however, RHS calls AMAT, and then the product of the matrices **A** and **Φ** is performed in RHS and thus, is outlined here.

$$\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi} \tag{3.6}$$

$$\Longrightarrow \begin{bmatrix} \dot{\phi}_{11} & \dot{\phi}_{12} \\ \dot{\phi}_{21} & \dot{\phi}_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \tag{3.7}$$

Thus,

$$\begin{aligned} \dot{\phi}_{11} &= (a_{11}\phi_{11} + a_{12}\phi_{21}) \\ \dot{\phi}_{12} &= (a_{11}\phi_{12} + a_{12}\phi_{22}) \\ \dot{\phi}_{21} &= (a_{21}\phi_{11} + a_{22}\phi_{21}) \\ \dot{\phi}_{22} &= (a_{21}\phi_{12} + a_{22}\phi_{22}) \end{aligned} \tag{3.8}$$

Equations 3.8 are evaluated in the subroutine RHS.

The last thing that RHS does is to take the instantaneous time-average of the logarithms of the eigenvalues by dividing $\ln \lambda_i$ by the instantaneous t. The actual calculation of the eigenvalues λ_i of the fundamental matrix **Φ** is performed in the

subroutine CALEIG which is described in section 3.4, and which is called by RHS.

3.3 Finding the Associated Linear System (AMAT)

In order to perform the integration of 3.2, the **A** matrix must first be computed. This relatively simple calculation is performed in the subroutine AMAT according to the following development.

Recall **A**(t) was defined as a Jacobian matrix: the matrix of partial derivatives of the system differential equations with respect to the variables of the problem. Therefore,

$$\mathbf{A}(t) = \begin{bmatrix} \frac{\partial x_2}{\partial x_1} & \frac{\partial x_2}{\partial x_2} \\ \frac{\partial(-bx_1 - ax_2)}{\partial x_1} & \frac{\partial(-bx_1 - ax_2)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ (-b - x_1 \frac{\partial b}{\partial x_1}) & (-a - x_2 \frac{\partial a}{\partial x2}) \end{bmatrix} \tag{3.9}$$

The subroutine RHS calls AMAT just prior to calculating the product $\mathbf{A\Phi}$.

3.4 Eigenvalues of the Fundamental Matrix (CALEIG)

The eigenvalues of a 2 × 2 fundamental matrix are easily calculated from its characteristic equation, which is defined by:

$$\det\{\mathbf{\Phi} - \lambda\mathbf{I}\} = \begin{vmatrix} \phi_{11} - \lambda & \phi_{12} \\ \phi_{21} & \phi_{22} - \lambda \end{vmatrix} = 0 \tag{3.10}$$

Evaluating the determinant yields

$$[(\phi_{11} - \lambda)(\phi_{22} - \lambda) - \phi_{12}\phi_{21}] = 0 \tag{3.11}$$

or,

$$\lambda^2 + \lambda(-\phi_{11} + \phi_{22}) + (\phi_{11}\phi_{22} - \phi_{12}\phi_{21}) = 0 \tag{3.12}$$

Solving the quadratic produces the following equation for λ:

$$\lambda = \frac{(\phi_{11} + \phi_{22}) \pm \sqrt{(-\phi_{11} - \phi_{22})^2 - 4(\phi_{11}\phi_{22} - \phi_{12}\phi_{21})}}{2} \tag{3.13}$$

In addition to calculating 3.13, the subroutine CALEIG computes the complex logarithm of the eigenvalues and returns them to RHS separated into their real and imaginary parts. To implement this separation, one needs to evaluate the natural logarithm of a complex number. Let the following equation define an arbitrary complex number:

$$\begin{aligned} z &= \alpha + i\beta \\ &= re^{i\theta} \text{ (polar form)} \end{aligned}$$

where,

$$r = |z| = \sqrt{\alpha^2 + \beta^2} \tag{3.14}$$

and

$$\theta = \arg z = \arctan\left(\frac{\beta}{\alpha}\right)$$

Taking the logarithm of z in its polar form yields

$$\begin{aligned} \ln(re^{i\theta}) &= \ln r + \ln e^{i\theta} \\ &= \ln r + i\theta \end{aligned} \tag{3.15}$$

Thus, the real part of $\ln z$ is $\ln r$, and the imaginary part is θ.

These are the values returned to RHS by CALEIG. RHS then divides the natural logarithms of the eigenvalues by t to yield the instantaneous rates of change.

3.5 Potential Problems

As was alluded to in the introductory part of this chapter, there are situations when the calculation of the eigenvalues of the fundamental matrix runs into problems. One of these problems is the fact that the eigenvalues are growing exponentially, and thus can quickly span several orders of magnitude. As this occurs, floating point overflow or underflow can result in the calculations.

The second problem of significance that can, and does, arise is eigenvalue bifurcation: two real eigenvalues merging into two complex, or two complex merging into two reals. Figure 3.1 shows a typical bifurcation. Typically, there will be an endless succession of these eigenvalue bifurcations, with the time between bifurcations

decreasing rapidly.

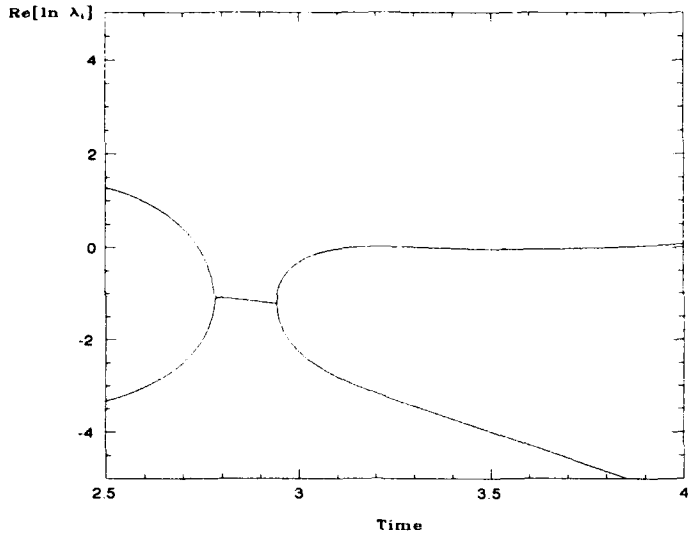

Figure 3.1. Sample Bifurcation

First, the software must be able to *detect* that an eigenvalue bifurcation has occurred. The bifurcation point is relatively straightforward to find for second-order systems, because the discriminant of the characteristic equation will change *sign* when a bifurcation occurs. Thus, comparison of the last discriminant calculated with the present one will detect whether or not a bifurcation has occurred. The subroutine SPLIT performs this comparison.

If a bifurcation has occurred, then the software must integrate through the bifurcation, and label/identify the eigenvalues as they merge. This is performed in the subroutine SPLIT2, which is called from SPLIT. Figure 3.1 illustrates that the slopes of the eigenvalues are infinite at the time of bifurcation. Defining a new time variable $s = |t - t_b|^{\frac{1}{2}}$, where t_b is the bifurcation time, eliminates this difficulty. The bifurcation time is calculated in SPLIT2 by noting that time is a

locally quadratic function of the difference between the two merging eigenvalues (real or imaginary parts as appropriate). Thus, the time of bifurcation is determined by inverse interpolation back to the time of the bifurcation. With the bifurcation time known, the merging eigenvalues are determined by extrapolating in the s variable to $s = 0$. (Polynomial extrapolation in time is impossible because of the infinite slopes seen in Figure 3.1.) Further extrapolation, slightly away from $s = 0$ (still using the s variable), enables the numerical integration to be restarted. This entire process introduces only minimal errors, and occurs within one normal integration timestep.

It is essential that the software keep track of which eigenvalue is which during a bifurcation so that characterization of their long term behavior is not compromised. Thus, SPLIT2 determines the bifurcation type (real-to-imaginary, or imaginary-to-real), and *tracks* the eigenvalues according to this type. If the *type* is determined to be a pair of reals going to complex, SPLIT2 insures that the root with the positive imaginary part emerges from the bifurcation with a positive slope. Since the two merging eigenvalues lose their identity in the bifurcation, SPLIT2 is free to swap root *labels* if necessary. If the *type* is a pair of complex values merging to real, SPLIT2 calls the subroutine PHASE to reset the 2π multipliers of the eigenvalues. It is only after this labelling, or tracking, has occurred that SPLIT2 extrapolates the average eigenvalues over the bifurcation.

The PHASE subroutine mentioned previously is also called from the main program to update the phase of the eigenvalues as they are propagated forward.

This ensures that the imaginary part of ln (λ_i) will have a net increase.

An additional check on the eigenvalues for second order systems is the conjugality of the eigenvalues. In the subroutine CHECKS, the imaginary parts of the eigenvalues and their averaged integrals are checked for *pairedness.* Specifically, the absolute value of the sum of the imaginary parts should be zero, within numerical accuracy.

The algorithms implemented to address these problems will be successful, if the growth of the eigenvalues stays reasonable during the time of integration, and the bifurcations do not occur back-to-back during a single timestep. If the growth is too rapid, accuracy is lost, and if back-to-back bifurcations occur, the bifurcations will not even be detected. In either of these cases, characterization of the long term behavior will be compromised.

IV. Application and Validation

Application of the technique, developed in previous chapters, to the van der Pol equation is offered here. The first sections contain the setup and analysis of different trial cases. A Fourier analysis, used for validation of the imaginary parts of the extended Lyapunov exponents, is presented in the last section.

4.1 Van der Pol's equation

Van der Pol's equation is a second-order differential equation with a linear restoring force and nonlinear damping, that originally arose as an idealization of a self-excited, or spontaneously oscillating, valve circuit(5). Per Jordan and Smith (5), the van der Pol equation may be written as

$$\ddot{x} + \epsilon(x^2 - 1)\dot{x} + x = 0 \tag{4.1}$$

Figures 4.1 and 4.2 contain the phase portraits for the van der Pol equation's two stable configurations. For $\epsilon < 0$, the van der Pol equation has a stable equilibrium point at the origin. As seen in Figure 4.1, all nearby trajectories spiral in toward the origin. Figure 4.2 illustrates that, with an $\epsilon > 0$, the van der Pol equation exhibits a limit cycle behavior, and is considered the classic paradigm for self-excited oscillations(7).

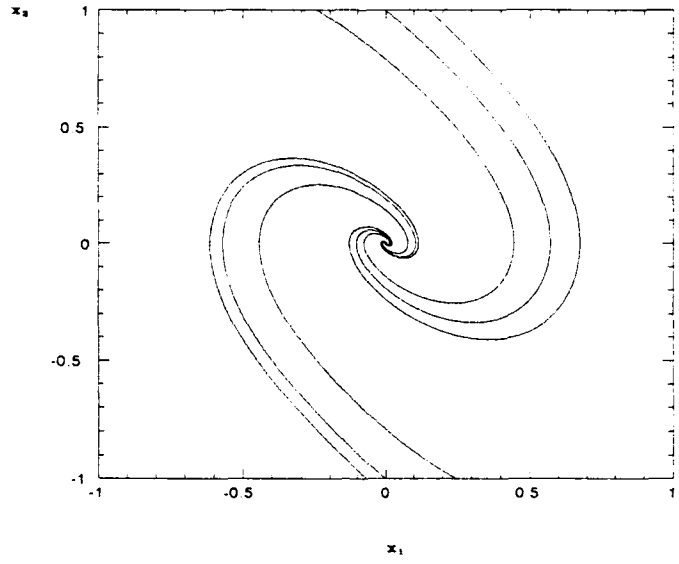

Figure 4.1. Phase Portrait for van der Pol's Equation ($\epsilon = -1$)

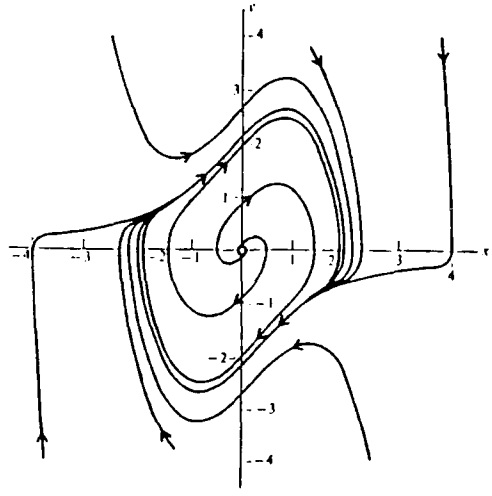

Figure 4.2. Phase Portrait for van der Pol's Equation ($\epsilon = 1$)

4.1.1 Setup First, this must be put into the general form $\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X}, t)$. To do this, let

$$\begin{aligned} x_1 &= x \\ x_2 &= \dot{x}_1 \end{aligned}$$

which yields,

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \text{and} & \\ \dot{x}_2 &= -\epsilon(x_1^2 - 1)x_2 - x_1 \end{aligned} \tag{4.2}$$

These are the equations required for the subroutine RHS.

Next, the equations for calculating the **A** matrix must be setup. Recall,

$$\mathbf{A} = \frac{\partial \mathbf{F}}{\partial \mathbf{X}} \tag{4.3}$$

Evaluating equation 4.3 produces the following expressions for the elements of **A**:

$$\begin{aligned} a_{11} &= \frac{\partial x_2}{\partial x_1} = 0 \\ a_{12} &= \frac{\partial x_2}{\partial x_2} = 1 \\ a_{21} &= \frac{\partial \left[-\epsilon(x_1^2 - 1)x_2 - x_1\right]}{\partial x_1} = -2\epsilon x_1 x_2 - 1 \end{aligned} \tag{4.4}$$

$$a_{22} = \frac{\partial[-\epsilon(x_1^2 - 1)x_2 - x_1]}{\partial x_2} = -\epsilon(x_1^2 - 1)$$

These are the equations that will be coded into the subroutine AMAT.

4.1.2 Case 1 (The Constant Coefficient Case) Linearizing about the equilibrium point, which is located at the origin for the van der Pol equation, yields an **A** matrix of constant coefficients for the equation $\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi}$, and the eigenvalues of **A** should correspond to the calculated extended Lyapunov exponents. Then the **A** matrix at $t = 0$ is equal to the **A** matrix at all time:

$$\mathbf{A}(t) = \mathbf{A}(t_0) = \begin{bmatrix} 0 & 1 \\ -1 & \epsilon \end{bmatrix} \tag{4.5}$$

Solving for the eigenvalues of **A**

$$\det\{\mathbf{A} - \lambda\mathbf{I}\} = \det\begin{bmatrix} 0-\lambda & 1 \\ -1 & \epsilon-\lambda \end{bmatrix} = 0 \tag{4.6}$$

$$= \lambda^2 - \epsilon\lambda + 1 = 0 \tag{4.7}$$

Solving equation 4.6 yields

$$\lambda = \frac{\epsilon \pm \sqrt{\epsilon^2 - 4}}{2} \tag{4.8}$$

From equation 4.8, it is abvious that choosing $|\epsilon|$ small (i.e. $|\epsilon| < 2$) will ensure an imaginary part of the eigenvalue. For this analysis, $\epsilon = -1$ will be used. Therefore,

$$\begin{aligned} \lambda &= \frac{-1 \pm \sqrt{1-4}}{2} \\ & -0.5 \pm \mathrm{i}\frac{\sqrt{3}}{2} \\ & -0.5 \pm (0.866)\mathrm{i} \end{aligned} \tag{4.9}$$

Figure 4.3 contains the phase portrait. Note that the equilibrium point is stable, and the trajectory is a spiral towards the point (0,0). Jordan and Smith's (5) treatment of the van der Pol equation contains a proof that the requirement for stability of the equilibrium point is a $\epsilon < 0$. Since $\epsilon = -1$ was chosen, the phase portrait shows that the system behaves in the expected fashion.

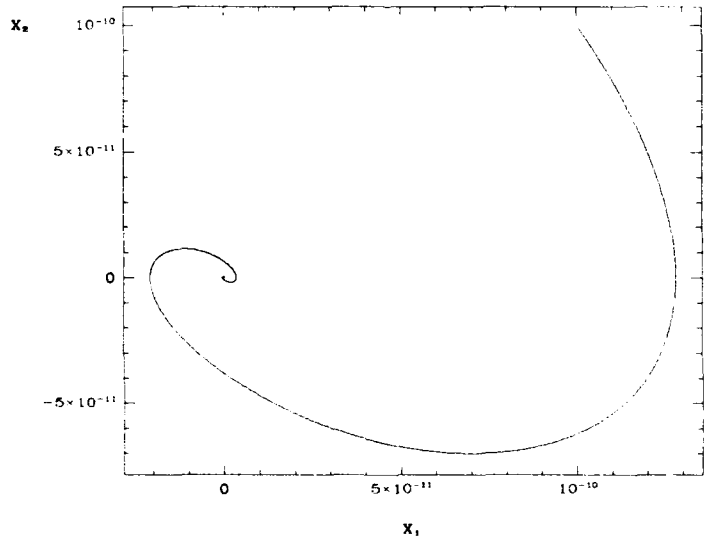

Figure 4.3. Van der Pol Equation Phase Portrait (Stable Equilibrium Point Case)

Figures 4.4 and 4.5 show the calculated real parts of the $\ln \lambda_i$ versus time. As expected, the logarithms of the eigenvalues are changing linearly with time, with a slope of -0.5.

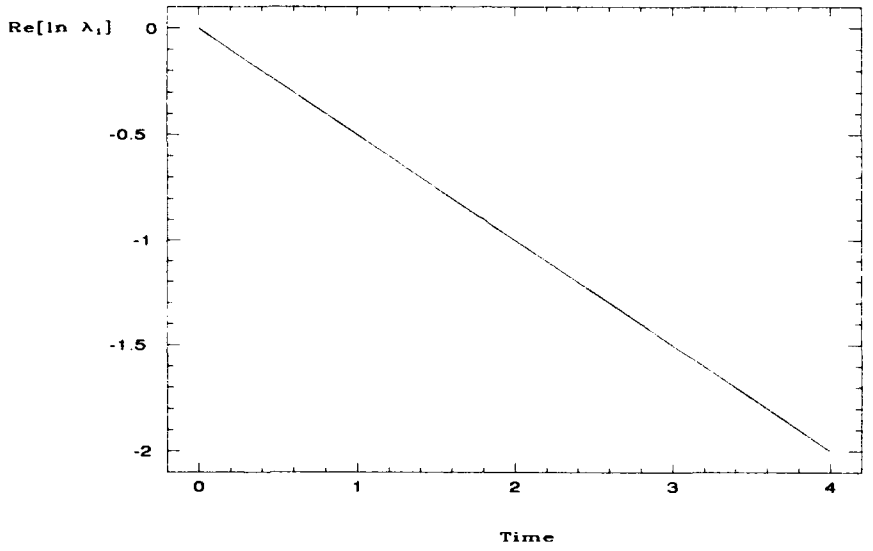

Figure 4.4. Real Part of $\ln \lambda_1$ versus Time

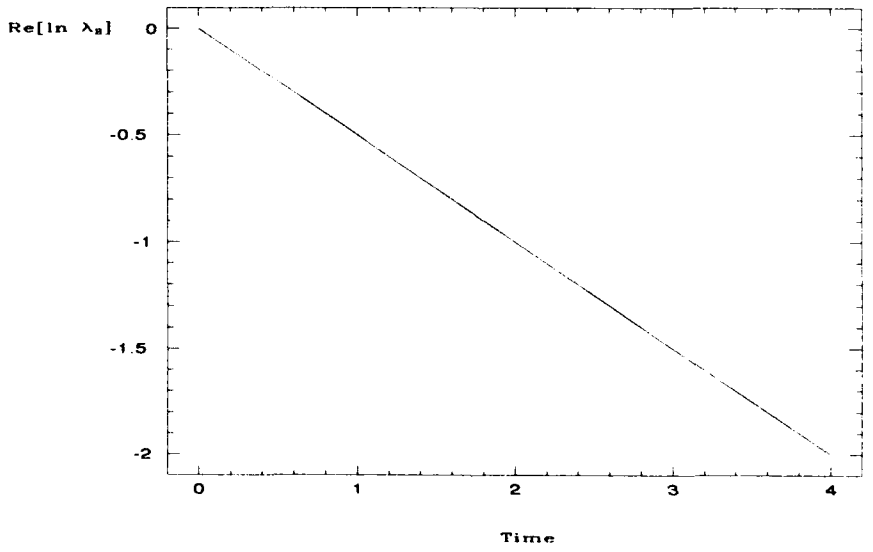

Figure 4.5. Real Part of $\ln \lambda_2$ versus Time

Figure 4.6 shows the calculated imaginary parts of $\ln \lambda_i$ versus time. The imaginary parts are changing linearly with time (slope$\approx$ 0.866), until t $\approx$ 3.6. At this time, the $\ln \lambda_i$ undergo back-to-back bifurcations that cannot be detected by the software.

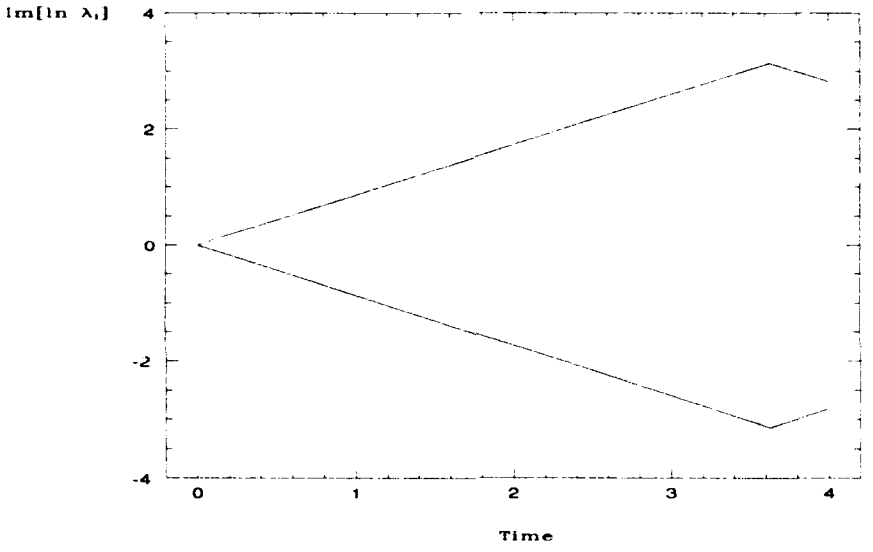

Figure 4.6. Imaginary Parts of $\ln \lambda_i$ versus Time

Figure 4.7 is a plot of the characteristic equation's discriminant versus time. This plot illustrates why there is a problem at $t \approx 3.6$. For, at this time, the back-to-back bifurcations occur so close together that the discriminant never appears to even change sign. Thus, the software never detects that a bifucation has occurred. This situation did not improve even when the program was run at the smallest timestep that was possible.

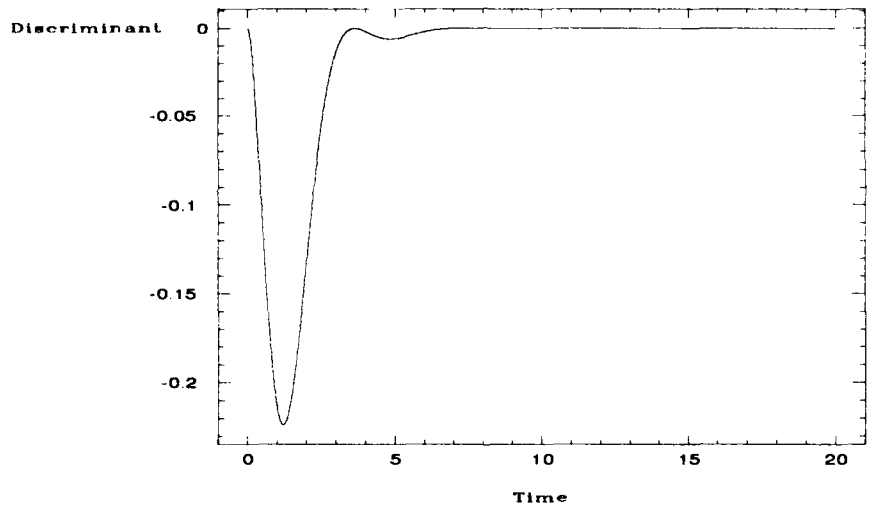

Figure 4.7. Discriminant of the Characteristic Equation

Thus, numerical inaccuracies force the integration to be limited to before the principal bifurcation. It is obvious though from figures 4.7 and 4.8, a plot of one of the variables versus time, that the solution quickly converges to a *steady-state* value in this case. Also, as the following figures will indicate, excellent results were still obtainable.

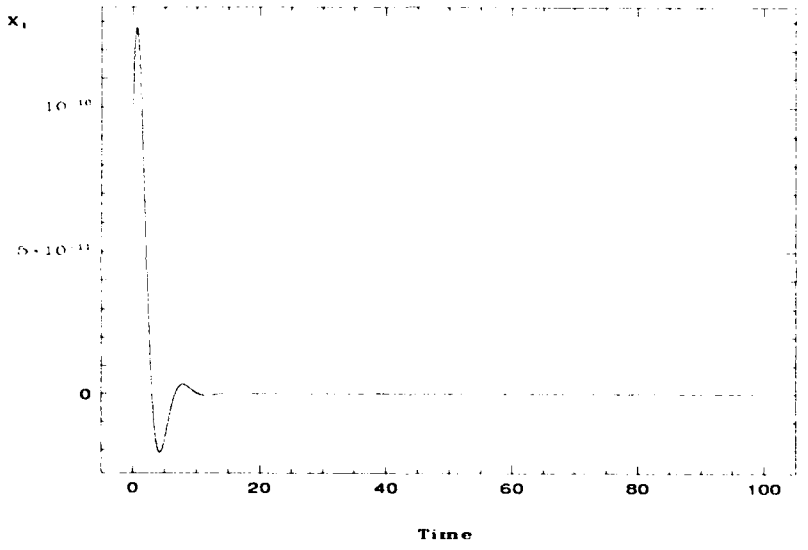

Figure 4.8. X_1 versus Time

Figures 4.9 and 4.10 contain plots of the real parts of the calculated extended Lyapunov exponents. Clearly, even with the bifurcation problem that was encountered, they are converging to the values obtained for the real parts of the eigenvalues of **A**: -0.5 (Reference equation 4.9).

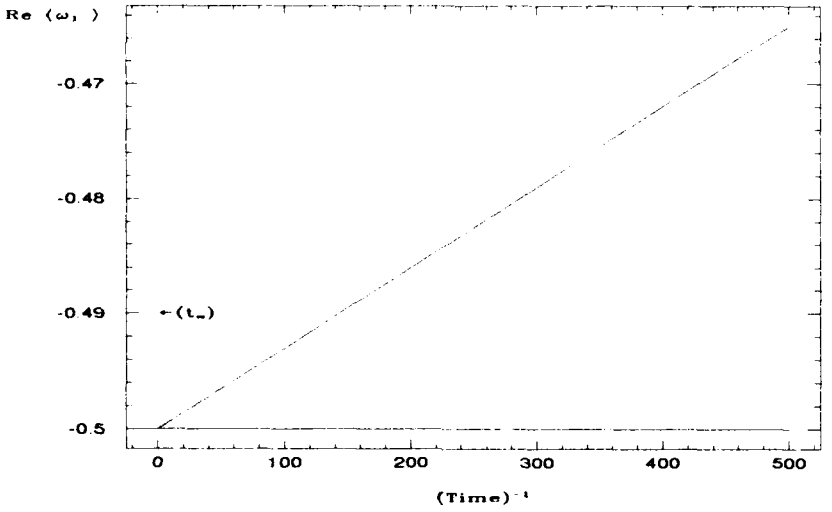

Figure 4.9. Real Part of *Extended* Lyapunov Exponent 1

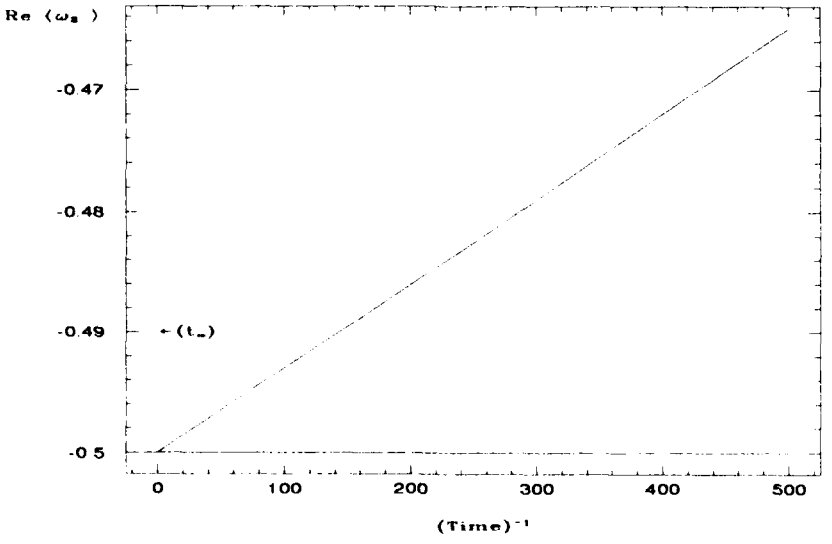

Figure 4.10. Real Part of *Extended* Lyapunov Exponent 2

Figure 4.11, the plot of the imaginary parts of the extended Lyapunov exponents, again shows that the technique has yielded the correct values. Clearly, as $t \rightarrow \infty$, the imaginary parts are converging to some number close to ± 0.86. This is the same value that was calculated for the imaginary parts of the eigenvalues of the constant coefficient **A** matrix.

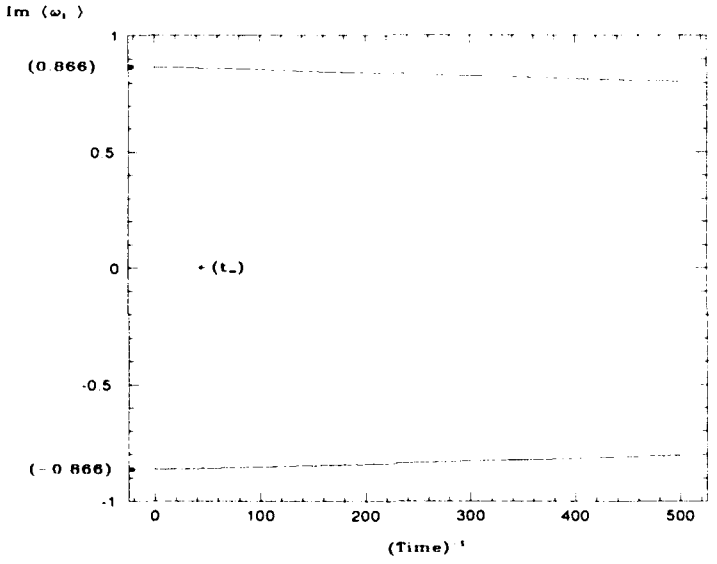

Figure 4.11. Imaginary Parts of *Extended* Lyapunov Exponents

It should be noted here that because the data was suspect after the principal bifurcation, it was not included for the extended Lyapunov exponent plots. However, the plots do conclusively show that the technique has yielded the correct values for the system characteristic exponents.

4.1.3 Case 2 (The Periodic Coefficient Case) Linearizing about the limit cycle yields an **A** matrix of periodic coefficients for the equation $\dot{\mathbf{\Phi}} = \mathbf{A}\mathbf{\Phi}$. Thus, the Poincaré exponents for this case should correspond to the extended Lyapunov exponents calculated using the technique outlined in the thesis. Per Jordan and Smith (5), the limit cycle, for $\epsilon = 1$, can be defined by the following initial conditions:

$$x_1(t_0) \approx 2 \tag{4.10}$$

$$x_2(t_0) \approx 0 \tag{4.11}$$

The angular frequency of the limit cycle(5) is

$$\omega = \frac{2\pi}{\tau} \tag{4.12}$$

where τ is the period of the limit cycle.

According to Floquet theory, the Poincaré exponents are defined as

$$p_i = \frac{1}{\tau} \ln \lambda_i(\tau) \pm \frac{i(2\pi)}{\tau} \tag{4.13}$$

where τ is the period of the trajectory and $\lambda_i(\tau)$ are the eigenvalues of the fundamental matrix $\mathbf{\Phi}$ at $t = \tau$. Thus, in order to calculate the Poincaré exponents, the period of the limit cycle must be determined, and then the eigenvalue of the fundamental matrix at the end of one period in time must be calculated.

From Figure 4.12, which is the plot of x_1 versus time, it can be seen that the period is equal to approximately 6.67.

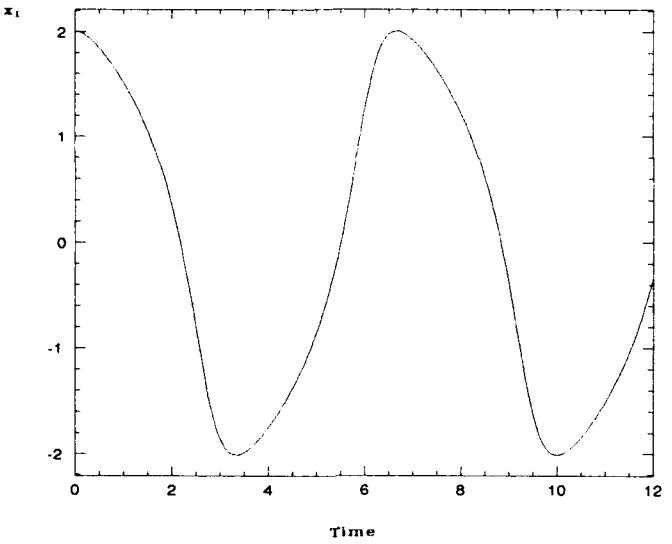

Figure 4.12. X_1 versus Time

Integration of the fundamental matrix $\mathbf{\Phi}$ through this time gave the following values for $\mathbf{\Phi}(\tau)$:

$$\mathbf{\Phi}(\tau) = \begin{bmatrix} 0.01998446583557 & 2.685067523504 \\ 0.006971779004636 & 0.9788204972462 \end{bmatrix} \tag{4.14}$$

A restatement of equation 3.13, which is used to calculate the eigenvalues of the fundamental matrix follows:

$$\lambda = \frac{(\phi_{11} + \phi_{22}) \pm \sqrt{(-\phi_{11} - \phi_{22})^2 - 4(\phi_{11}\phi_{22} - \phi_{12}\phi_{21})}}{2} \tag{4.15}$$

Evaluating the above using the elements of the $\mathbf{\Phi}$ matrix from equation 4.14 yields the following values for $\lambda_i(\tau)$

$$\lambda_1(\tau) = 0.997962 \tag{4.16}$$

$$\lambda_2(\tau) = .000843226 \tag{4.17}$$

$$\tag{4.18}$$

The Poincaré exponents are then calculated using these values for $\lambda_i(\tau)$ in equation 4.13, and they are found to be

$$p_1 = \frac{\ln \lambda_1}{\tau} \approx 0 \tag{4.19}$$

$$p_2 = \frac{\ln \lambda_2}{\tau} \approx -1.05 \tag{4.20}$$

The reader should note that the $\pm i2\pi/\tau$ term of equation 4.13 has been dropped. This is standard practice in Floquet theory because this term is the uncertainty that arises due to the limits on the natural logarithm function.

Figure 4.13 contains the phase portrait which was plotted with the system elements calculated using the technique outlined in the previous chapter, for the following initial conditions and system parameters:

$$x_1 = 2.0$$

$$x_2 = 0.0$$

$$\epsilon = +1$$

Obviously, the trajectory is the limit cycle given in Jordan and Smith (5).

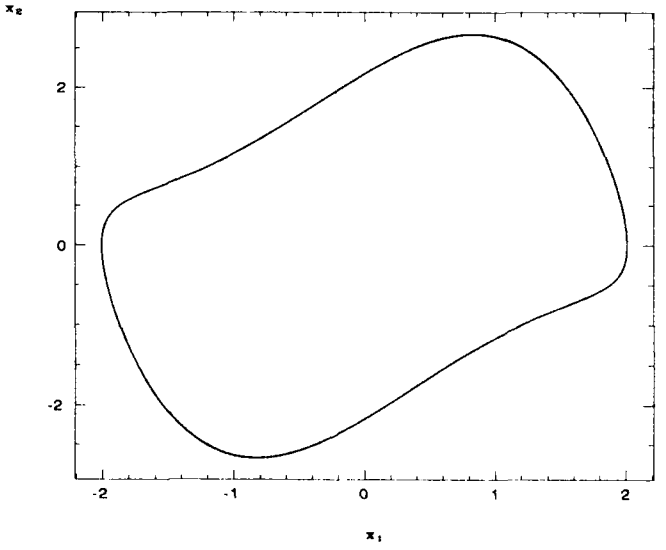

Figure 4.13. Van der Pol Equation Phase Portrait (Limit Cycle Case)

Figure 4.14 contains the plot of the real parts of $\ln \lambda_i$ versus time. As expected, the logarithms are changing linearly with time, and engage in a continual series of bifurcations. These bifurcations begin to cause problems at about t = 6 because of the small amount of time between bifurcations. Additionally, it can be seen that the eigenvalues themselves are now spread over at least five orders of magnitude, with the smallest being very close to zero. Thus, accuracy is compromised at the end of the integration time.

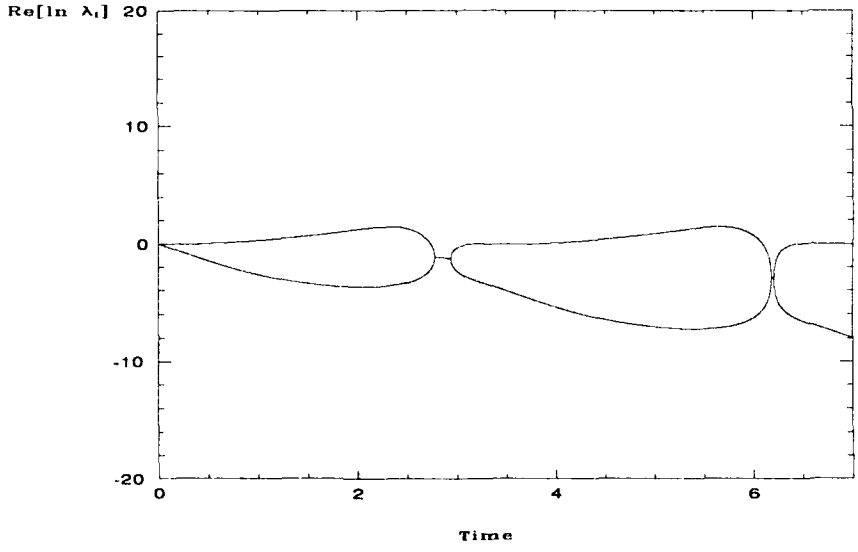

Figure 4.14. Real Parts of $\ln \lambda_i$ versus Time

Figure 4.15 shows the calculated imaginary parts of the $\ln \lambda_i$ versus time. The imaginary parts show a characteristic (11) "stairstep" behavior with the phase angle continually increasing, on the average. Recall, the identity of each root is chosen at the time of bifurcation to ensure that this occurs. This is absolutely necessary if the long term behavior of the system is to be characterized.

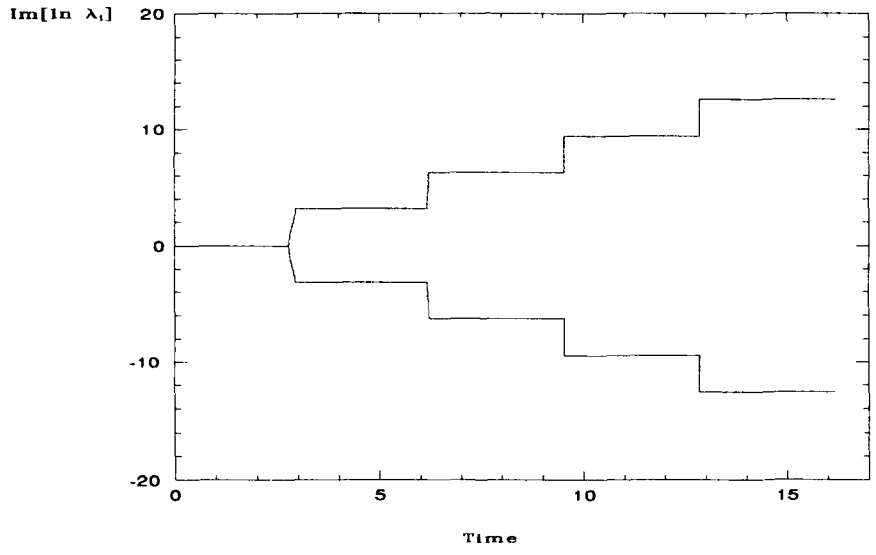

Figure 4.15. Imaginary Parts of $\ln \lambda_i$ versus Time

For comparison's sake, Figures 4.16 and 4.17 are plots of the discriminant of the characteristic equation for this case. It is obvious from 4.16 that numerous bifurcations are ocurring and that after the principal bifucation, they are ocurring closer in time.

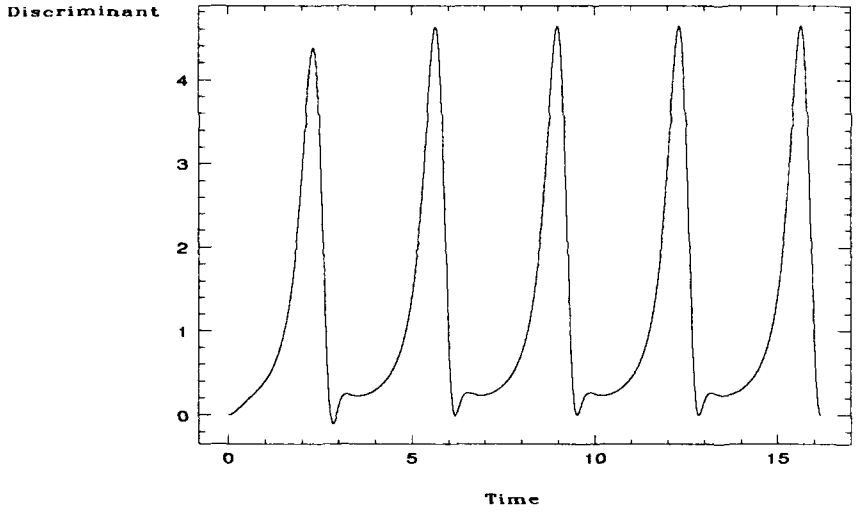

Figure 4.16. Discriminant of the Characteristic Equation

Figure 4.17 shows that, especially for the principal bifurcation, the discriminant changes sign, and remains negative for a time. This implies that there is not a back-to-back bifurcation at the principal bifurcation time. Thus, the software detected the bifurcation and successfully tracked the eigenvalues through the bifurcation, unlike the bifurcations in the equilibrium point case.

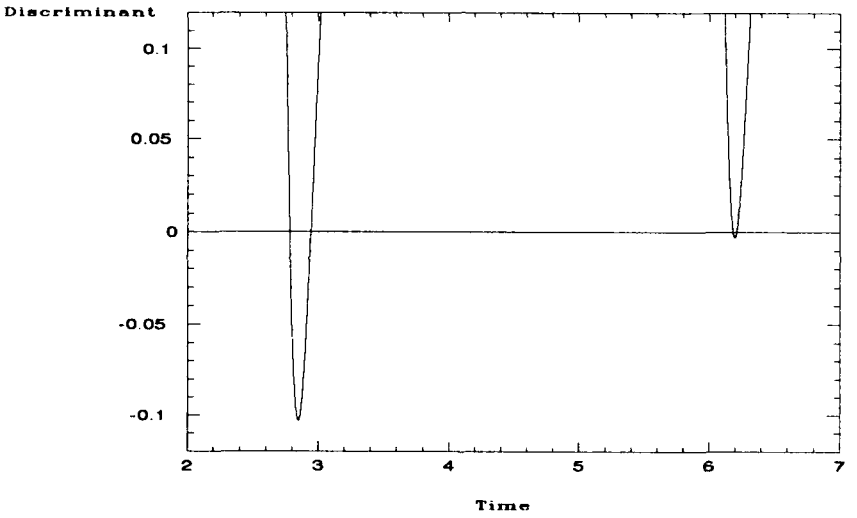

Figure 4.17. Discriminant of the Characteristic Equation

The real parts of the extended Lyapunov exponents versus inverse time are plotted in Figures 2.3 and 2.4. Extrapolating out to where inverse time is zero (i.e., $t \rightarrow \infty$), the real parts are obviously converging to the real parts of the Poincaré exponents: 0 and -1.05.

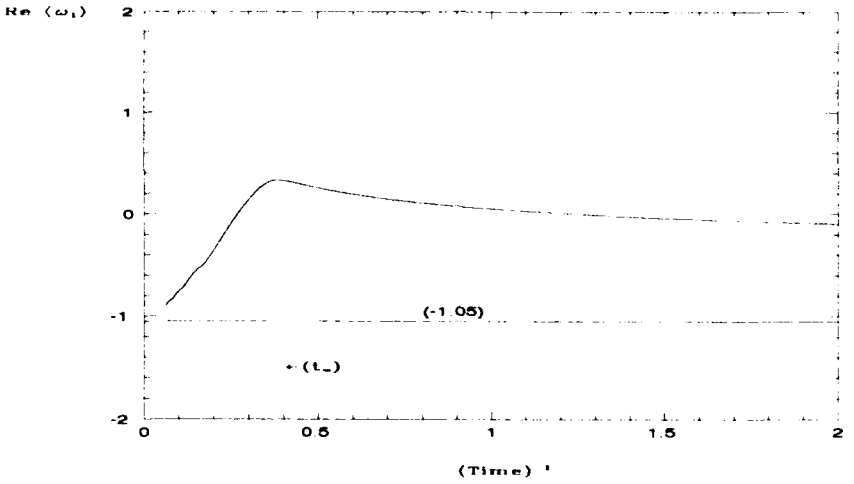

Figure 4.18. Real Part of *Extended* Lyapunov Exponent 1

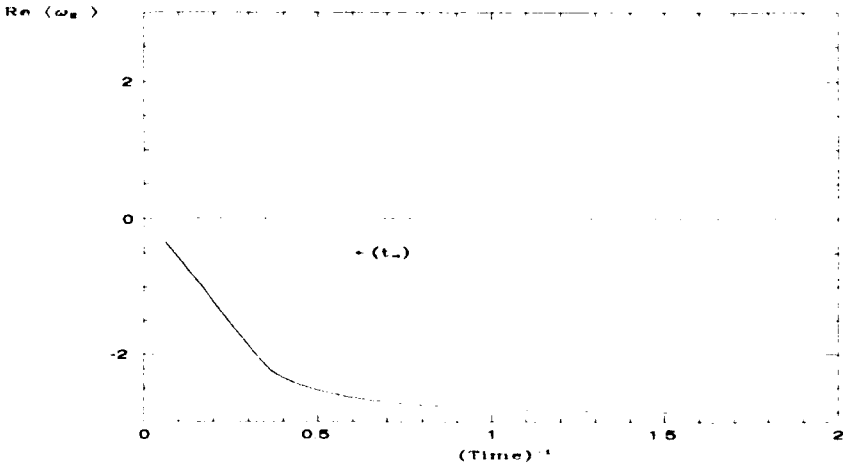

Figure 4.19. Real Part of *Extended* Lyapunov Exponent 2

The imaginary parts of the extended Lyapunov exponents are plotted in Figure 4.20 below. Although calculation of the Poincaré exponents did not yield an imaginary part, Figure 4.20 confirms that there is an imaginary part. Though the extended Lyapunov exponents start out at zero, as inverse time approaches zero ($t \rightarrow \infty$), the imaginary parts are converging to $\omega_i \approx \pm 0.9$, which is approximately equal to the "uncertainty" term in the equation for calculating the Poincaré exponents: $\pm\frac{i(2\pi)}{\tau} = \pm\frac{i(2\pi)}{6.67} = 0.94$.

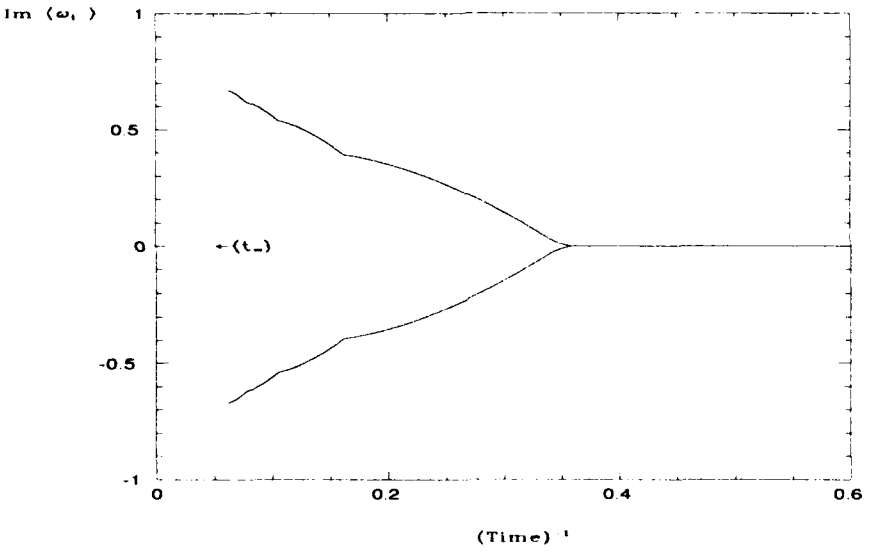

Figure 4.20. Imaginary Parts of *Extended* Lyapunov Exponents

Thus, although standard Floquet theory does not yield an imaginary part, the technique outlined in this paper confirms such a part exists. Further validation of existence, and correctness of the values obtained for the imaginary parts is presented in the next section.

4.2 Validation of the Imaginary Part of the Extended *Lyapunov Exponent*

To validate the imaginary part of the extended Lyapunov exponent, FFT (fast Fourier transform) techniques(12) will be used to obtain the PSD (power spectral density) plot for the system in each of the trial cases. Appendix B contains the computer code used for this analysis. Most often, waveforms/data are measured in the time domain. Time-based measurements have historical precedence. For example, Galileo reportedly used his own pulse as a timepiece in making his original pendulum observations. Unquestionably, time-based measurements are the most familiar data format, but time histories tell only one side of the story. The *type* of power vs frequency spectrum that a system possesses will classify its response. For instance, if the spectrum contains n-number of spikes at discrete frequencies, it is considered a periodic response of period n, and if the spectrum is continuous, with no discernible spikes, it is classified chaotic. Additionally, if there are spikes in the spectrum, these spikes will occur at the driving frequencies of the system.

4.2.1 Application of FFT Techniques For each of the trial cases, the trajectories were numerically integrated using 4.2, and a single *displaced* trajectory on the coordinatized tangent space was also integrated using

$$\delta\dot{\mathbf{X}}(t) = \mathbf{A}(t)\delta\mathbf{X}(t) \tag{4.21}$$

At frequent, equally spaced time increments, the displacement vector $\delta\mathbf{X}(t)$ was normalized, coordinate values were saved, and the integration was restarted. Given a random initial condition for $\delta\mathbf{X}(t_0)$, it is expected that the trajectory will strongly converge towards the extended Lyapunov exponent with the largest real part. Normalizing the vector $\delta\mathbf{X}(t)$ discards the exponential growth information, and retains only information on how the vector rotates in space (i.e., it retains the information on the imaginary part). Then, individual coordinate histories were processed with the fast Fourier transform to extract frequency information. The subsequent one sided power spectral density will show whether the relative motion has a discrete or continuous frequency spectrum, and if there are discrete spikes, these will occur at the fundamental frequency of the system.

4.2.2 Validation of the van der Pol Equilibrium Point Case Application of this technique to the van der Pol equilibrium point case produced Figure 4.21. Clearly, the spectrum is discrete, with a large spike at angular frequency approximately 0.9 and a much smaller peak at three times this value.

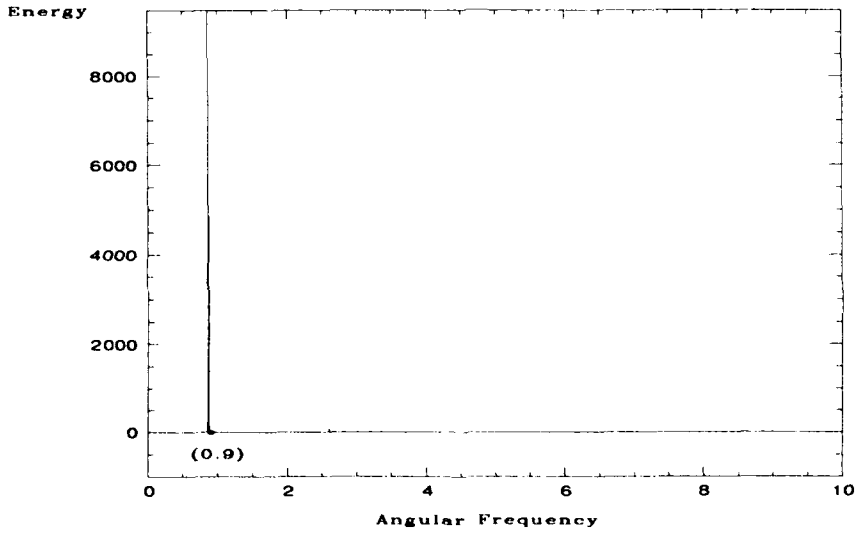

Figure 4.21. Power Spectral Density of a Relative Trajectory versus Angular Frequency

Figure 4.22 is an expanded view of the portion of the spectrum close to the largest spike, and shows that the large spike actually occurs at something less than 0.9, about 0.86.

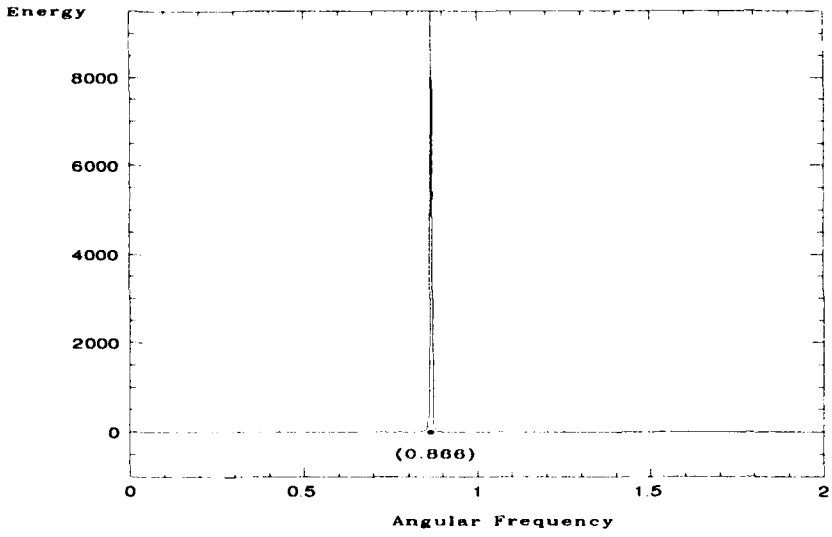

Figure 4.22. Power Spectral Density of a Relative Trajectory versus Angular Frequency

Thus, the spike occurs at exactly the value obtained for the imaginary part of the extended Lyapunov exponent (≈ 0.86), and the imaginary part of the eigenvalues of the **A** matrix (0.86).

4.2.3 Validation of the van der Pol Limit Cycle Case Application of the fast Fourier transform technique to the van der Pol limit cycle case produced Figure 4.23. Again the spectrum is discrete with a large spike at angular frequency approximately 0.9 and a much smaller peak at three times this value.

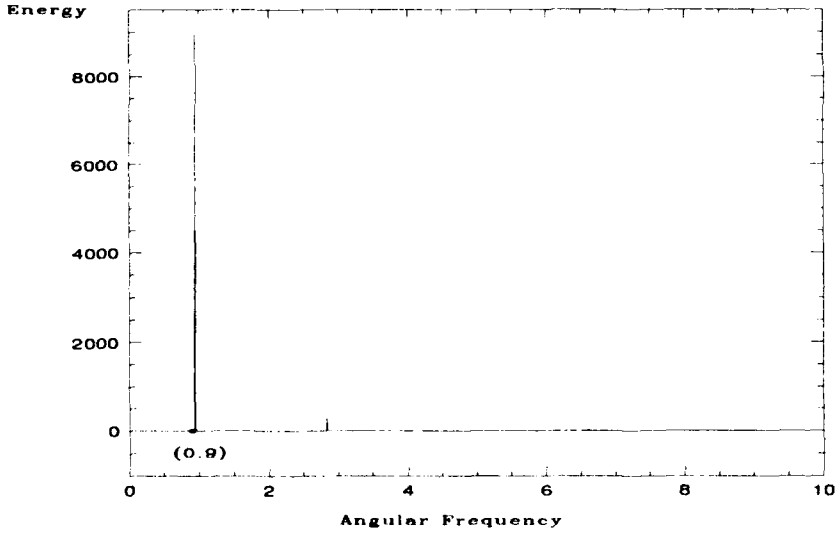

Figure 4.23. Power Spectral Density of a Relative Trajectory versus Angular Frequency

Figure 4.24 is an expanded view of the portion of the spectrum close to the largest spike, and shows that the large spike actually occurs at something greater than 0.9, about 0.94.

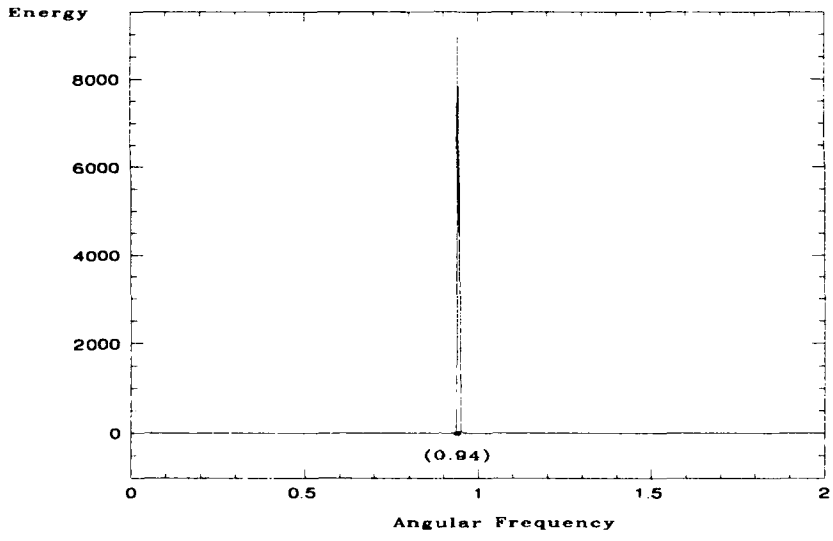

Figure 4.24. Power Spectral Density of a Relative Trajectory versus Angular Frequency

Again, the value obtained for the imaginary part of the extended Lyapunov exponent is validated. Additionally, this plot serves as further confirmation that an imaginary part of the Poincaré exponents exists, and is equal to

$$\pm\frac{i2\pi}{\tau}$$

V. *Conclusions and Recommendations*

An explicit numerical technique has been developed to calculate the characteristic exponents (*extended* Lyapunov exponents) for any second order system. The fact that this technique can be applied to *any* system unifies the theory of second-order linear systems. A summary of conclusions, and recommendations for further investigation are presented in the following sections.

5.1 Conclusions

The extended Lyapunov exponents calculated for the van der Pol equation stable equilibrium point case were in excellent agreement with the theory: they exactly equalled the eigenvalues of the constant coefficient **A** matrix. Additionally, the discrete spike in the PSD at the value of the imaginary part of the extended Lyapunov exponent for this system clearly validated the imaginary part of the exponent as being the system's fundamental frequency.

The value of the extended Lyapunov exponent for the periodic case (the van der Pol equation limit cycle) went beyond simple agreement, it confirmed the existence of an imaginary part to the Poincaré exponent, **and** determined its value, a question that Floquet theory left unanswered in this case. Also, the clear spike in the PSD at the same value as the imaginary part of the extended Lyapunov exponent further validated its value and existence.

These results suggest that the potential for unifying linear system theory in general is very great. The concept that a system's characteristic exponents contain both a real and imaginary part, whether the system is constant, periodic, multiply periodic, or aperiodic, is not only intuitively correct, it has been shown to be numerically correct for the trial cases presented here.

5.2 Recommendations

Recommendations for further study of the extended Lyapunov exponent and applications of this technique are:

1) Apply the technique to the van der Pol system with a periodic forcing function.

2) Apply the technique to other classic second-order systems (e.g., the pendulum).

3) Develop the code and apply the technique to some classic higher order systems.

The major difficulty anticipated with these recommendations has to do with the calculation of the eigenvalues of the fundamental matrix $\mathbf{\Phi}$. The timespan over which this technique is applicable is currently limited by pairs of eigenvalue bifurcations that are separated by short, exponentially decreasing time intervals. In his original paper on extended Lyapunov exponents (11), Wiesel developed an algorithm for propagating the eigenvalue and eigenvector matrices numerically. In essence, it was

shown that the eigenvectors and eigenvalues obey differential equations of their own, thereby eliminating the need to propagate the fundamental matrix itself. Use of this algorithm could possibly reduce the eigenvalue bifurcation difficulty. Additionally, for higher order systems, direct calculation of the eigenvalues of the fundamental matrix $\mathbf{\Phi}$ becomes much messier than the second order systems; thus, numerically integrating the eigenvalue and eigenvector differential equations themselves will most likely be more efficient and accurate in the application of the technique to higher order systems.

Appendix A. *Computer Programs for Calculating the Extended Lyapunov Exponents*

```
c
      program vp1main
c
c     2D simple van der Pol system
c     calculate eigenvalues from phi
c     and time averaged eigenvalues
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig
      double precision disc1,dtbif
      common /debug/ idebug
      double precision tmax,omegr,omegi
      double precision a(2,2),phi(2,2),work(40),wpp(2),vmag
      double precision wqq(2),wrr(2),wss(2)
      complex*16 wq,wr,ws
      complex*16 w(2),vec(2,2),wp
      equivalence( wp,wpp),(wq,wqq),(wr,wrr),(ws,wss)
c
c     input parameters, Ic's
c
      read (*,*) t
      read (*,*) e
c
c     open output file
c
      open(13,FILE='run.out',STATUS='UNKNOWN')
C PRODUCTION CODE
      OPEN(11,FILE='lyap.re',STATUS='UNKNOWN')
      OPEN(12,FILE='lyap.im',STATUS='UNKNOWN')
C END PRODUCTION CODE
C CHECKOUT CODE
      OPEN(1,FILE='lneig1.re',STATUS='UNKNOWN')
      OPEN(2,FILE='lneig2.re',STATUS='UNKNOWN')
      OPEN(3,FILE='lneig1.im',STATUS='UNKNOWN')
      OPEN(4,FILE='lneig2.im',STATUS='UNKNOWN')
      OPEN(7,FILE='IMSL1.re',STATUS='UNKNOWN')
      OPEN(8,FILE='IMSL2.re',STATUS='UNKNOWN')
      OPEN(9,FILE='IMSL1.im',STATUS='UNKNOWN')
      OPEN(10,FILE='IMSL2.im',STATUS='UNKNOWN')
      OPEN(20,FILE='disc1',STATUS='UNKNOWN')
      OPEN(23,FILE='phase',STATUS='UNKNOWN')
C END CHECKOUT CODE
c
```

```
c
c     initialize at t=0, read if t not zero.......
c
      if(t .eq. 0.d0) then
c
c          beginning a run......read state
c
           read (*,*) x(1,1),x(2,1)
c
c          initialize Phi = I, integrals of eigenvlaues to zero
c
           do 100 i = 3,10
                x(i,1) = 0.d0
  100      continue
           do 101 i = 3,6,3
                x(i,1) = 1.d0
  101      continue
           do 102 i = 1,2
               eig(i,1) = (0.d0, 0.d0)
               npi(i) = 0
  102      continue
c          initialize swap buffer
           do 103 i = 1,2
                iswap(i) = i
  103      continue
c          mode 1, no bif checking at first
           mode = 1
      else
c
c          resuming a run....
c
           do 104 i = 1,9,2
                read (*,*) x(i,1),x(i+1,1)
  104      continue
c          read discriminants
           read (*,*) disc1(4)
c          read last eigenvalues
           do 105 i = 1,4
                read (*,*) wpp(1),wpp(2)
                eig(i,4) = wp
  105      continue
           read (*,*) imode
           read (*,*) (npi(jj),jj=1,2)
           read (*,*) (iswap(jj),jj=1,2)
c          begin in normal mode
```

```
          mode = 0
      endif
      read (*,*) tmax,nstp,nskp
      read (*,*) idebug
c
      nn =  10
      hh = (tmax-t)/(dble(nstp*nskp))
      nxt = 0
      call haming(nxt)
C DEBUG
c     write (13,*) 'eig before phase call'
c     do 254 i = 1,2
c         write (13,*) 'eig',i
c             do 254 j = 1,2
c                 write (13,*) j,eig(i,j)
c 254 continue
C END
      if(nxt .eq. 0) stop 77
c     adjust startup phases
      do 29 i = 2,4
          call phase(i)
   29 continue
C DEBUG
c     write (13,*) 'eig after phase calls'
c     do 255 i = 1,2
c          write (13,*) 'eig',i
c          do 255 j = 1,2
c                write (13,*) j,eig(i,j)
c 255 continue
C END
      write (13,*) 'haming initialized'
c     cycle 5 times before bifurcation checking
      do 30 i = 1,5
          call haming(nxt)
          call phase(nxt)
   30 continue
c     turn on bifurcation checking
      mode = 0
c
c     open output files
c
c
c
c     integration loop
c
```

```
      do 1000 istp = 1,nstp
c
c          do eigenvalue calc every other time....
c
         if(mod(istp,1) .eq. 0) then
           do 500 i = 1,2
                do 500 j = 1,2
                    phi(i,j) = x(2*j+i,nxt)
  500      continue
c
c          call eigrf(phi,2,2,1,w,vec,2,work,ier)
c
c          the eigrf subroutine was not in the IMSL library for
c          eigenvalue computation, but devcrg is
c
           call devcrg(2,phi,2,w,vec,2)
c
           do 600 i = 1,2
                wp = w(i)
c               calc complex log w(i)
c
                if(t .eq. 0.d0) go to 600
                omegr = dlog(dsqrt( wpp(1)*wpp(1) +
     1                      wpp(2)*wpp(2) ))
                omegi = datan2( wpp(2) , wpp(1) )
c
c     the following if statement is to write the log mag and phase of the
c     IMSL generated eigenvalues  in the same order as our literal method
c     (i.e. the calceig output):  largest to smallest, or positive complex
c     to negative complex
c
                  if(i.eq.1)then
                   write (8,9) t,omegr
                   write (10,9) t,omegi
                  elseif(i.eq.2)then
                   write (7,9) t,omegr
                   write (9,9) t,omegi
                  endif
  600      continue
         endif
    9      format(2x,2(e20.13,1x))
C CHECKOUT CODE:  write the "literal" method log mag and phase into
c                 file eig1(or 2).re(im)
           do 700 i = 1,2
               wp = eig(i,nxt)
```

```
                write (i,9) t,wpp(1)
                write (i+2,9) t,wpp(2)
  700       continue
C END CHECKOUT CODE

c           write eigenvalues from phi by "literal" method
c           over t, as a function of inverse time
c
C PRODUCTION CODE
            if(t .ne. 0.d0) then
    8           format(4(1x,e18.10),/,1x,e18.10)
                write (11,8) 1.d0/t,x(7,nxt)/t,x(8,nxt)/t
                write (12,8) 1.d0/t,x(9,nxt)/t,x(10,nxt)/t
            endif
C END PRODUCTION CODE
c
c           do the integration
c
            do 900 jskp = 1,nskp
c
                 call haming(nxt)
c
                 call split(nxt)
c
c                adjust phase after bif checks
                 call phase(nxt)
c
  900       continue
c
 1000 continue
c
c     dump final state
c
      call dump(nxt)
c
      stop
      end

c$INCLUDE: 'haming.for'
c$INCLUDE: 'rhs_vp1.for'
c$INCLUDE: 'calceig_vp1.for'
c$INCLUDE: 'split_vp1.for'
c$INCLUDE: 'split2_vp1.for'
c$INCLUDE: 'checks_vp1.for'
c$INCLUDE: 'dump_vp1.for'
```

```
c$INCLUDE: 'phase_vp1.for'
c$INCLUDE: 'amat_vp1.for'
c
      include 'haming_vp1.for'
      include 'rhs_vp1.for'
      include 'calceig_vp1.for'
      include 'split_vp1.for'
      include 'split2_vp1.for'
      include 'checks_vp1.for'
      include 'dump_vp1.for'
      include 'phase_vp1.for'
      include 'amat_vp1.for'
```

```
c
      subroutine haming(nxt)
c
c     haming is an ordinary differential equations integrator
c     it is a fourth order predictor-corrector algorithm
c     which means that it carries along the last four
c     values of the state vector, and extrapolates these
c     values to obtain the next value (the prediction part)
c     and then corrects the extrapolated value to find a
c     new value for the state vector.
c
c     the value nxt in the call specifies which of the 4 values
c     of the state vector is the "next" one.
c     nxt is updated by haming automatically, and is zero on
c     the first call
c
c     the user supplies an external routine rhs(nxt) which
c     evaluates the equations of motion
c
      common /ham/ x,y(10,4),f(10,4),errest(10),n,h,mode,e
      double precision x,y,f,errest,h,hh,xo,tol,e
c
c     all of the good stuff is in this common block.
c     x is the independent variable ( time )
c     y(6,4) is the state vector- 4 copies of it, with nxt
c                 pointing at the next one
c     f(6,4) are the equations of motion, again four copies
c                 a call to rhs(nxt) updates an entry in f
c     errest is an estimate of the truncation error - normally not
c                 used
c     n is the number of equations being integrated - 6 or 42 here
c     h is the time step
c     mode is 0 for just EOM, 1 for both EOM and EOV
c
      tol = 0.0000000001d+00
c     switch on starting algorithm or normal propagation
      if(nxt) 190,10,200
c
c     this is hamings starting algorithm....a predictor - corrector
c     needs 4 values of the state vector, and you only have one- the
c     initial conditions.
c     haming uses a Picard iteration (slow and painfull) to get the
c     other three.
c     if it fails, nxt will still be zero upon exit, otherwise
c     nxt will be 1, and you are all set to go
```

```
c
   10 xo = x
      hh = h/2.0d+00
      call rhs(1)
      do 40 l = 2,4
      x = x + hh
      do 20 i = 1,n
   20 y(i,l) = y(i,l-1) + hh*f(i,l-1)
      call rhs(l)
      x = x + hh
      do 30 i = 1,n
   30 y(i,l) = y(i,l-1) + h*f(i,l)
   40 call rhs(l)
      jsw = -20
   50 isw = 1
      do 120 i = 1,n
      hh = y(i,1) + h*( 9.0d+00*f(i,1) + 19.0d+00*f(i,2)
     1     - 5.0d+00*f(i,3) + f(i,4) ) / 24.0d+00
      if( dabs( hh - y(i,2)) .lt. tol ) go to 70
      isw = 0
   70 y(i,2) = hh
      hh = y(i,1) + h*( f(i,1) + 4.0d+00*f(i,2) + f(i,3))/3.0d+00
      if( dabs( hh-y(i,3)) .lt. tol ) go to 90
      isw = 0
   90 y(i,3) = hh
      hh = y(i,1) + h*( 3.0d+00*f(i,1) + 9.0d+00*f(i,2) + 9.0d+00*f(i,3)
     1     + 3.0d+00*f(i,4) ) / 8.0d+00
      if( dabs(hh-y(i,4)) .lt. tol ) go to 110
      isw = 0
  110 y(i,4) = hh
  120 continue
      x = xo
      do 130 l = 2,4
      x = x + h
  130 call rhs(l)
      if(isw) 140,140,150
  140 jsw = jsw + 1
      if(jsw) 50,280,280
  150 x = xo
      isw = 1
      jsw = 1
      do 160 i = 1,n
  160 errest(i) = 0.0
      nxt = 1
      go to 280
```

```
  190 jsw = 2
      nxt = iabs(nxt)
c
c     this is hamings normal propagation loop -
c
  200 x = x + h
      np1 = mod(nxt,4) + 1
      go to (210,230),isw
c     permute the index nxt modulo 4
  210 go to (270,270,270,220),nxt
  220 isw = 2
  230 nm2 = mod(np1,4) + 1
      nm1 = mod(nm2,4) + 1
      npo = mod(nm1,4) + 1
c
c     this is the predictor part
c
      do 240 i = 1,n
      f(i,nm2) = y(i,np1) + 4.0d+00*h*( 2.0d+00*f(i,npo) - f(i,nm1)
     1     + 2.0d+00*f(i,nm2) ) / 3.0d+00
  240 y(i,np1) = f(i,nm2) - 0.925619835d0*errest(i)
c
c     now the corrector - fix up the extrapolated state
c     based on the better value of the equations of motion
c
      call rhs(np1)
      do 250 i = 1,n
      y(i,np1) = ( 9.0d+00*y(i,npo) - y(i,nm2) + 3.0d+00*h*( f(i,np1)
     1     + 2.0d+00*f(i,npo) - f(i,nm1) ) ) / 8.0d+00
      errest(i) = f(i,nm2) - y(i,np1)
  250 y(i,np1) = y(i,np1) + 0.0743801653d0 * errest(i)
      go to (260,270),jsw
  260 call rhs(np1)
  270 nxt = np1
  280 return
      end
```

```
c
c
      subroutine rhs(nxt)
c
c     Van der Pol's "simple" equation
c     eom and phi matrix, integrals of eigenvalues dt
c
      common /ham/t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig,wp
      double precision disc1,dtbif,wpp(2)
      common /debug/ idebug
      double precision a(2,2)
      double precision xx(2)
      equivalence (wp,wpp)
c
C DEBUG
c     write (*,*) 'rhs, nxt=',nxt
c     write (*,*) 'timestep', hh
c     write (*,*) 'full state/rate dump at entry'
c     do 333 i = 1,nn
c          write (*,*) i,x(i,nxt),f(i,nxt)
c 333 continue
C END
c
c     extract state
c
      do 1 i = 1,2
           xx(i) = x(i,nxt)
    1 continue
c
c     Van der Pohl's eom
c
      f(1,nxt) = x(2,nxt)
      f(2,nxt) =-x(1,nxt)+(e-e*x(1,nxt)*x(1,nxt))*x(2,nxt)
c
c     A(t) matrix
c
      call amat(nxt,a)
c
c     phi dot = a phi
c     phi stored by cols
c
      do 50 i = 1,2
```

```
          do 50 j = 1,2
              f(i+2*j,nxt) = 0.d0
              do 50 k = 1,2
                   f(i+2*j,nxt) = f(i+2*j,nxt) + a(i,k)*x(k+2*j,nxt)
   50 continue
c
c     calculate eigenvalues
c
      if( t .ne. 0) then
           call caleig(nxt)
      endif
c
c     eigenvalue average eom
c
      if( t .ne. 0) then
           wp = eig(1,nxt)
           f(7,nxt) = wpp(1)/t
           f(9,nxt) = wpp(2)/t
           wp = eig(2,nxt)
           f(8,nxt) = wpp(1)/t
           f(10,nxt) = wpp(2)/t
      else
           f(7,nxt) = 0.d0
           f(8,nxt) = 0.d0
           f(9,nxt) = 0.d0
           f(10,nxt) = 0.d0
      endif
c
C DEBUG
c     if(idebug .ge. 5) then
c          write (13,*) 'rhs, full state/rate dump, nxt=',nxt
c          do 666 i = 1,10
c               write (13,*) x(i,nxt),f(i,nxt)
c 666      continue
c      endif
C END
c
      return
      end
```

```
c
c
      subroutine amat(nxt,a)
c
c     calculate A matrix for Van der Pol's 'simple ' system.
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      double precision xx(2),a(2,2)
c
c
c     extract state
c
      do 5 i = 1,2
           xx(i) = x(i,nxt)
    5 continue
c
c
       a(1,1)=0.d0
       a(1,2)=1.d0
       a(2,1)=-1.d0-2.d0*e*xx(1)*xx(2)
       a(2,2)=e-e*xx(1)*xx(1)
c
      return
      end
```

```
c
c
      subroutine caleig(nxt)
c
c     calculate eigenvalues of Phi matrix for simple 2D van der Pol
c     system, from integrated Phi matrix elements
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig
      double precision disc1,dtbif,a,b,c
      complex*16 fig(4),ca,cb
      common /debug/ idebug
c
      double precision phi(2,2),wpp(2),wqq(2),vmag
      double precision twopi
      complex*16 wp,wq,cdisc
      equivalence (wp,wpp),(wq,wqq)
C DEBUG
c     if(idebug .ge. 2) write (13,*) 'enter calceig'
C END
c
      twopi = 2.d0*3.141592653589793d0
      nm1 = nxt + 3
      if(nm1 .gt. 4) nm1 = nm1 - 4
c
c     extract phi, stored by cols
c
      do 10 i = 1,2
          do 10 j = 1,2
               phi(i,j) = x(2*j+i,nxt)
   10 continue
c
c      solve for eigenvalues of phi (2x2), and discriminant (disc1)
c      disc1 is the argument under the square root in the quadratic eq
c      solution.  Therefore, it indicates whether we have a real, or
c      complex eigenvalue.
c
       b = -phi(1,1)-phi(2,2)
       c = (phi(1,1)*phi(2,2)) - (phi(1,2)*phi(2,1))
c
C DEBUG
c     if(idebug .ge. 2) write (13,*) 'A',b
C END
```

```
c
      disc1(nxt) = .25d0*b*b - c
c
c     note: the following write statement to file 20, called disc, is
c     used to plot the discriminant vs time.  This enables us to see
c     the long term behavior of the discriminant,  and therefore correct
c     for bifurcations
c
      write(20,20) t,disc1(nxt)
20    format(2x,2(e20.13,1x))
c
c
c
      if (disc1(nxt) .ge. 0.d0) then
          wpp(1) = dsqrt (disc1(nxt))
          wpp(2) = 0.d0
      else
          wpp(1) = 0.d0
          wpp(2) = dsqrt (dabs (disc1(nxt) ) )
      endif
c
      wqq(1) = -b/2.d0
      wqq(2) = 0.d0
c
C DEBUG
c     if(idebug .ge. 2) then
c               write (13,*) 'disc',disc1(nxt)
c               write (13,*) 'sqrt',wp
c     endif
C ENDIF
c
      fig(1) = wq + wp
      fig(2) = wq - wp
c
c     iswap will keep track of bifurcation "direction"
c
      do 200 i = 1,2
          eig(i,nxt) = fig(iswap(i))
200   continue
c
c     calculate logs of eigenvalues
c
      do 100 i = 1,2
           wp = eig(i,nxt)
           wqq(1) = dlog( dsqrt( wpp(1)*wpp(1) + wpp(2)*wpp(2) ))
```

```
            wqq(2) = datan2( wpp(2), wpp(1) ) + dble(npi(i))*twopi
C DEBUG
c            if(idebug .ge. 2) then
c                write (13,*) 't=',t
c                write (13,*) 'i, lambda',i,eig(i,nxt)
c                write (13,*) 'i, ln lam',i,wq
c             endif
C END
            eig(i,nxt) = wq
  100 continue
c
C DEBUG
c     if(idebug .ge. 2) write (13,*) 'exit calceig'
C END
      return
      end
```

```
c
c
      subroutine split(nxt)
c
c     check for root bifurcations and fix up results
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig,etemp1,etemp2
      double precision disc1,dtbif,dtemp1,dtemp2
      common /debug/ idebug
c
      double precision wpp(2),wqq(2)
      complex*16 wp,wq
      equivalence (wp,wpp),(wq,wqq)
c
c     do bifurcation checks? not on startup
c
      if(mode .ne. 0) return
c
c
c     BIFURCATION CHECKS!
c
      nm1 = nxt + 3
      if(nm1 .gt. 4) nm1 = nm1 - 4
c
c     check for isolated bifurcation
c
      if( disc1(nxt)*disc1(nm1) .lt. 0.d0 ) then
c
c DEBUG
c           if(idebug .gt. 0) then
c                write (13,*) 'isolated bif'
c                write (13,*) 'real pts',wpp(1),wqq(1)
c           endif
c END
            call split2(nxt)
c
c           do reasonableness checks, write breakpt file
c
C DEBUG     temp higher debug level....
            ida = idebug
            if(idebug .ge. 1) idebug = 2
            if(ida .gt. 0) then
```

```
                write (13,*) 'post split2 checks'
              endif
C END
              call checks(nxt)
C DEBUG       reset debug level
              idebug = ida
C END
c
c             bifurcation introduces a discontinuity in integral slopes
c             haming must be restarted
c
              do 200 i = 1,10
                   x(i,1) = x(i,nxt)
  200         continue
              etemp1 = eig(1,nm1)
              etemp2 = eig(1,nxt)
              eig(1,1) = etemp2
              eig(1,4) = etemp1
              etemp1 = eig(2,nm1)
              etemp2 = eig(2,nxt)
              eig(2,1) = etemp2
              eig(2,4) = etemp1
              dtemp1 = disc1(nm1)
              dtemp2 = disc1(nxt)
              disc1(1) = dtemp2
              disc1(4) = dtemp1
              nxt = 0
C DEBUG
c             ida = idebug
c             idebug = 6
C END
              call haming(nxt)
              if(nxt .eq. 0) then
                write (13,*) 'haming restart failure after bifurcation'
                stop
              endif
C DEBUG
c             idebug = ida
C END
c             PRAY HERE THAT ANOTHER BIF DIDN'T OCCUR
c
      endif
c
C DEBUG
c     write (13,*) 'exit split'
```

```
C END
      return
      end
```

```
c
c
      subroutine split2(nxt)
c
c     handles isolated bifurcations of a single +- root pa
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig
      double precision disc1,dtbif
      common /debug/ idebug
      complex*16 wp,wq
      double precision dnxt,dnm1,dnm2,dnm3
      double precision wpp(2),wqq(2),twopi
      double precision p,dp,capd,capdot
      double precision sm1,sm2,sm3,cm1,cm2,cm3,ints(4),splus2,tbif
      complex*16 eig0(2)
      equivalence (wp,wpp),(wq,wqq)
c
      twopi = 2.d0*3.141592653589793d0
c
C DEBUG
      if(idebug .ne. 0) then
          write (13,*) ' '
          write (13,*) 'isolated bifurcation, root pair'
          write (13,*) 'near t=',t
      endif
C END
c
c     set indices to extract roots, discriminants
c
      nm1 = nxt + 3
      if(nm1 .gt. 4) nm1 = nm1 - 4
      nm2 = nxt + 2
      if(nm2 .gt. 4) nm2 = nm2 - 4
      nm3 = nxt + 1
      if(nm3 .gt. 4) nm3 = nm3 - 4
      i1 = 1
      i2 = 2
c
      dnxt = disc1(nxt)
      dnm1 = disc1(nm1)
      dnm2 = disc1(nm2)
      dnm3 = disc1(nm3)
```

```
C DEBUG
      if(idebug .ne. 0) then
          write (13,*) 'roots ',i1,i2
          write (13,*) 'dnxt',dnxt
          write (13,*) 'dnm1',dnm1
          write (13,*) 'dnm2',dnm2
          write (13,*) 'dnm3',dnm3
      endif
C END
c
c     determine bifurcation time
c
      p = -dnxt /( dnxt - dnm1 )
      do 50 i = 1,15
           capd = (p+1.d0)*(p+2.d0)*(p+3.d0)*dnxt/6.d0
     1           - p*(p+2.d0)*(p+3.d0)*dnm1/2.d0
     2           + p*(p+1.d0)*(p+3.d0)*dnm2/2.d0
     3           - p*(p+1.d0)*(p+2.d0)*dnm3/6.d0
           capdot = ((p+2.d0)*(p+3.d0)+(p+1.d0)*(p+3.d0)+
     1                (p+1.d0)*(p+2.d0))*dnxt/6.d0
     2           - ((p+2.d0)*(p+3.d0)+p*(p+3.d0)+p*(p+2.d0))
     3               * dnm1/2.d0
     4           + ((p+1.d0)*(p+3.d0)+p*(p+3.d0)+p*(p+1.d0))
     5               * dnm2/2.d0
     6           - ((p+1.d0)*(p+2.d0)+p*(p+2.d0)+p*(p+1.d0))
     7               * dnm3/6.d0
           dp = -capd/capdot
c DEBUG
           if(idebug .ne. 0) then
               write (13,*) 'D, Ddot, p, dp',capd,capdot,p,dp
           endif
c END
           p = p + dp
           if(dabs(dp) .lt. 1.d-7) go to 55
   50 continue
      write (13,*) 'SPLIT2: bifurcation time iteration failed'
      stop 63
   55 continue
      dtbif = p*hh
C DEBUG
      if(idebug .ne. 0) then
          write (13,*) 'dtbif',dtbif
      endif
C END
c
```

```
c     determine bifurcation type
c
      if(dnxt .lt. 0.d0) go to 2000
c
 1000 continue
c
c     ******************************************************
c     *                                                    *
c     *  current discriminant positive, imag pair -> reals *
c     *                                                    *
c     ******************************************************
c
C DEBUG
      if(idebug .ne. 0) then
          write (13,*) 'imag -> real'
      endif
C END
c
      itype = 1
c
c     reset two pi multipliers and correct new roots
c
      call phase(nxt)
c
c     now fix up average eigenvalue integrals over bifurcation
c
      go to 5000
c
 2000 continue
c
c     ************************************************************
c     *                                                          *
c     *   current discriminant neg, real pair -> imaginaries     *
c     *                                                          *
c     ************************************************************
c
C DEBUG
      if(idebug .ne. 0) then
          write (13,*) 'real -> imag'
      endif
C END
c
      itype = 2
c
c     swap this pair as necessary to insure that root with positive
```

```
c     imag part always increases......
c
c     who has positive imag part? check last root
      ipos = i1
      wp = eig(i1,nm1)
      if(wpp(2) .lt. 0.d0) ipos = i2
c     bifurcation on left or right side of zero?  Check root phase
c     prior root is real, and prior phase is a multiple of pi...
      wp = eig(ipos,nm1)
c     reduce mod pi
      icyc = idnint( 2.d0*wpp(2)/twopi )
      iside = +1
      if( mod(icyc, 2) .ne. 0) iside = -1
c     reduce NEW root phase mod 2 pi to see if increasing or decreasing
      wp = eig(ipos,nxt)
      jcyc = idint( wpp(2)/twopi )
      wpp(2) = wpp(2) - dble(jcyc)*twopi
C DEBUG
      if(idebug .gt. 0) then
           write (13,*) 'positive root now',ipos,eig(ipos,nxt)
           write (13,*) 'positive root pre',ipos,eig(ipos,nm1)
           write (13,*) 'prior phase / pi',icyc
           write (13,*) 'new reduced phase',wpp(2)
           write (13,*) 'axis side',iside
      endif
C END
c     determine if swapping needed...side dependent
      if(iside .gt. 0) then
           if(wpp(2) .gt. 4.71d0) go to 2005
      else
           if(wpp(2) .lt. 3.141592653589d0) go to 2005
      endif
c     no swapping needed
C DEBUG
      if(idebug .ne. 0) write (13,*) 'no swapping needed'
C END
      go to 2010
 2005 continue
c
c     swap this root pair!
c
      ineg = i1
      if(ineg .eq. ipos) ineg = i2
      wp = eig(ipos,nxt)
      wpp(2) = wpp(2) + (dble(npi(ineg)) - dble(npi(ipos)))*twopi
```

```
      wq = eig(ineg,nxt)
      wqq(2) = wqq(2) + (dble(npi(ipos)) - dble(npi(ineg)))*twopi
      eig(ipos,nxt) = wq
      eig(ineg,nxt) = wp
c     swap iswap order....its tied to calceig order
      isw = iswap(ipos)
      iswap(ipos) = iswap(ineg)
      iswap(ineg) = isw
C DEBUG
      if(idebug .ne. 0) then
           write (13,*) 'roots swapped'
      endif
C END
c
 2010 continue
c
c     reset two pi multipliers and correct new roots
c
      call phase(nxt)
c
c     ******************************************************
c     *      carry average eigenvalue ints over bif        *
c     ******************************************************
c
 5000 continue
C DEBUG
      if(idebug .ne. 0) then
            write (13,*) 'pre-bif eigenvalues'
            write (13,*) eig(1,nm1)
            write (13,*) eig(2,nm1)
            write (13,*) 'post-bif eigenvalues'
            write (13,*) eig(1,nxt)
            write (13,*) eig(2,nxt)
      endif
C END
c
c     extrapolate integrals, log eigenvalues into bif, s interp
c
c     s factors, (dtbif is negative)
      sm1 = dsqrt( dabs( hh + dtbif ))
      sm2 = dsqrt( dabs( hh + dtbif ) + hh )
      sm3 = dsqrt( dabs( hh + dtbif ) + 2.d0*hh )
c
c     extrap coef
      cm1 = sm2*sm3/( (sm1-sm2)*(sm1-sm3) )
```

```
      cm2 = sm1*sm3/( (sm2-sm1)*(sm2-sm3) )
      cm3 = sm1*sm2/( (sm3-sm1)*(sm3-sm2) )
c
      eig0(1) = cm1*eig(1,nm1) + cm2*eig(1,nm2) + cm3*eig(1,nm3)
      eig0(2) = cm1*eig(2,nm1) + cm2*eig(2,nm2) + cm3*eig(2,nm3)
c DEBUG
c     if(idebug .ne. 0) then
c          write (13,*) 'extrapolation factors'
c          write (13,*) cm1,cm2,cm3
c          write (13,*) 'eigenvalues extrapolated to s=0'
c          write (13,*) eig0(1)
c          write (13,*) eig0(2)
c     endif
C END
c
      ints(1) = cm1*x(7,nm1) + cm2*x(7,nm2) + cm3*x(7,nm3)
      ints(2) = cm1*x(8,nm1) + cm2*x(8,nm2) + cm3*x(8,nm3)
      ints(3) = cm1*x(9,nm1) + cm2*x(9,nm2) + cm3*x(9,nm3)
      ints(4) = cm1*x(10,nm1) + cm2*x(10,nm2) + cm3*x(10,nm3)
C DEBUG
c     if(idebug .ne. 0) then
c          write (13,*) 'last 3 pre bif integrals'
c          write (13,*) '7 ',x(7,nm1),x(7,nm2),x(7,nm3)
c          write (13,*) '8 ',x(8,nm1),x(8,nm2),x(8,nm3)
c          write (13,*) '9 ',x(9,nm1),x(9,nm2),x(9,nm3)
c          write (13,*) '10',x(10,nm1),x(10,nm2),x(10,nm3)
c          write (13,*) 'averaged integrals extrap to s=0'
c          write (13,*) ints(1),ints(3)
c          write (13,*) ints(2),ints(4)
c     endif
C END
c
c     integrate out of singularity, s integral, ln eigenvalues
c     assumed linear in s
c
      splus2 =  dabs( dtbif )
      tbif = t + dtbif
c
      wp = eig0(1)
      wq = eig(1,nxt)
      x(7,nxt) = ints(1) + 2.d0*splus2*( wpp(1)/6.d0 + wqq(1)/3.d0)/tbif
      x(9,nxt) = ints(3) + 2.d0*splus2*( wpp(2)/6.d0 + wqq(2)/3.d0)/tbif
      wp = eig0(2)
      wq = eig(2,nxt)
      x(8,nxt) = ints(2) + 2.d0*splus2*( wpp(1)/6.d0 + wqq(1)/3.d0)/tbif
```

```
      x(10,nxt)= ints(4) + 2.d0*splus2*( wpp(2)/6.d0 + wqq(2)/3.d0)/tbif
C DEBUG
c     if(idebug .ne. 0) then
c          write (13,*) 'ints extrap to end of timestep'
c          write (13,*) x(7,nxt),x(9,nxt)
c          write (13,*) x(8,nxt),x(10,nxt)
c     endif
C END
c
      return
      end
```

```
c
c
      subroutine phase(nxt)
c
c     fix phase on current eigenvalues, update npi
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig
      double precision disc1,dtbif
      common /debug/ idebug
      complex*16 wp,wq
      double precision wpp(2),wqq(2),twopi
      equivalence (wp,wpp),(wq,wqq)
c
      twopi = 2.d0*3.141592653589793d0
c
c
c     set indices to extract roots
c
      nm1 = nxt + 3
      if(nm1 .gt. 4) nm1 = nm1 - 4
c
c     reset two pi multipliers and correct new roots
c
      do 100 i1 = 1,2
           wp = eig(i1,nxt)
           wq = eig(i1,nm1)
           idel = nint( (wqq(2) - wpp(2))/twopi )
C DEBUG
           if(idebug .ge. 2) then
                write (23,*) 'npi',npi(i1)
                write (23,*) 'eig ',i1,' now',wp
                write (23,*) 'eig ',i1,' pre',wq
                write (23,*) 'idel',idel
           endif
C END  note:  file 23 is called 'phase'
c
           if(idel .ne. 0) then
                npi(i1) = npi(i1) + idel
                wpp(2) = wpp(2) + dble(idel)*twopi
                eig(i1,nxt) = wp
C DEBUG
                if(idebug .ge. 2) then
```

```
                write (23,*) 'corrected eig',wp
              endif
C END
          endif
  100 continue
c
      return
      end
```

```
c
c
      subroutine checks(nxt)
c
c     do some reasonableness checks for van der Pol's "simple system"
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig
      double precision disc1,dtbif
      common /debug/ idebug
c
      double precision wpp(2),wqq(2)
      complex*16 wp,wq
      equivalence (wp,wpp),(wq,wqq)
c
c     check eigenvalues  (imaginary parts) for pairedness
c
      do 100 i = 1,1
           wp = eig(2*(i-1)+1,nxt)
           wq = eig(2*(i-1)+2,nxt)
c          imag parts negs
           if(dabs(wpp(2)+wqq(2)) .gt. 0.1d0) then
                 itrip = 2
                 go to 666
           endif
  100 continue
c
c     check averaged integrals for pairedness
c
      if( dabs(x(9,nxt)+x(10,nxt)) .gt. 0.0005d0) then
            itrip = 4
            go to 666
      endif
c
c     we've escaped the big red button one more time
c     save here.....
c
      write (13,*) 'breakpoint file written at t=',t
      call dump(nxt)
c
      return
c
c     ABORT PROCESSING
```

```
c
  666 continue
c
      write (13,*) 'CHECK ABORT'
      write (13,*) 'at t=',t
c
      if(itrip .eq. 1) then
           write (13,*) 'real parts not negs'
      elseif(itrip .eq. 2) then
           write (13,*) 'imag parts not negs'
      elseif(itrip .eq. 3) then
           write (13,*) 'real parts integrals not neg'
      else
           write (13,*) 'imag parts integrals not neg'
      endif
c
      nm1 = nxt - 1
      if(nm1 .le. 0) nm1 = nm1 + 4
      nm2 = nxt - 2
      if(nm2 .le. 0) nm2 = nm2 + 4
      nm3 = nxt - 3
      if(nm3 .le. 0) nm3 = nm3 + 4
c
      do 10 j = 1,2
            write (13,*) 'eig',j
            write (13,*) '   nxt',eig(j,nxt)
            write (13,*) '   nm1',eig(j,nm1)
            write (13,*) '   nm2',eig(j,nm2)
            write (13,*) '   nm3',eig(j,nm3)
   10 continue
c
c
      write (13,*) 'disc1'
      write (13,*) '     nxt',disc1(nxt)
      write (13,*) '     nm1',disc1(nm1)
      write (13,*) '     nm2',disc1(nm2)
      write (13,*) '     nm3',disc1(nm3)
c
      stop
      end
```

```
c
      subroutine dump(nxt)
c
c     dump current state to breakpoint file
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig
      double precision disc1,dtbif
      common /debug/ idebug
      double precision wpp(2)
      complex*16 wp
      equivalence( wp,wpp)
c
c     dump current state to file
c
      open(15,FILE='break.pt')
c
  123 format(1x,e20.13,1x,e20.13)
      write (15,123) t
      do 124 i = 1,9,2
           write (15,123) x(i,nxt),x(i+1,nxt)
  124 continue
  125 format(1x,4(i5,1x))
      write (15,123) disc1(nxt)
      do 127 i = 1,2
           wp = eig(i,nxt)
           write (15,123) wpp(1),wpp(2)
  127 continue
      write (15,125) (npi(jj),jj=1,2)
      write (15,125) (iswap(jj),jj=1,2)
c
      close(15)
c
      return
      end
```

Appendix B. *Computer Programs Used for Fast Fourier Transform/Power Spectral Density Analysis*

```
c
      program fftmain
c
c     2D system
c     integrate the eom and variational equations for a 2D system
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /debug/ idebug
      double precision tmax,dmag
      double precision a(2,2),work(40)
c
c     input parameters, Ic's
c
      read (*,*) t
      read (*,*) e
c
c     open output file
c
      open(13,FILE='run.out',STATUS='UNKNOWN')
C PRODUCTION CODE
      OPEN(1,FILE='state',STATUS='UNKNOWN')
      OPEN(2,FILE='var1',STATUS='UNKNOWN')
      OPEN(3,FILE='var2',STATUS='UNKNOWN')
C END PRODUCTION CODE
c
c
c     initialize at t=0,
c
c          beginning a run......read state
c
           read (*,*) x(1,1),x(2,1),x(3,1),x(4,1)
c
c    read integration parameters:  max time, # of steps in max time,
c                                        how often integration is reported
      read (*,*) tmax,nstp,nskp
c
c          nn is the number of equations to be integrated, hh will determine
c          the time step that the integrator, haming, will use.  Four copies
c    of the state vector are kept around, thus the incrementor nxt.
c
      nn =  4
      hh = (tmax-t)/(dble(nstp*nskp))
      nxt = 0
      call haming(nxt)
```

```
c
      if(nxt .eq. 0) stop 77
      write (13,*) 'haming initialized'
c     integration loop
c
      do 1000 istp = 1,nstp
c


c
c      do the integration
c
          do 500 jskp = 1,nskp
c
               call haming(nxt)
c
c
  500      continue
c
c PRODUCTION CODE
c
c          normalize variation
         dmag=dsqrt(x(3,nxt)*x(3,nxt) + x(4,nxt)*x(4,nxt))
c
         do 600 i=1,2
          x(i+2,nxt)= x(i+2,nxt)/dmag
600      continue
c
c
  9      format(2x,5(e20.13,1x))
         write (1,9) t,x(1,nxt),x(2,nxt)
         write (2,9) t,x(3,nxt)
         write (3,9) t,x(4,nxt)
c
c       move state vector and variation vector into slot 1
c
      do 900 i=1,4
        x(i,1)=x(i,nxt)
900   continue
c
c       reinitialize haming

      nxt = 0
      call haming(nxt)
c
      if(nxt .eq. 0) stop 77
```

```
      write(13,*) t, 'haming initialized'
c
 1000 continue
c
c
c
      stop
      end

      include 'haming_vp.for'
      include 'fftrhs_vp.for'
      include 'amat_vp.for'
```

```
c
c
      subroutine rhs(nxt)
c
c     Van der Pol's  equation eom and equations of variation
c
      common /ham/t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      common /eval/ eig(2,4),npi(2),disc1(4),dtbif,iswap(2)
      complex*16 eig,wp
      double precision disc1,dtbif,wpp(2)
      common /debug/ idebug
      double precision a(2,2)
      double precision xx(2)
      equivalence (wp,wpp)
c
c     extract state and variation
c
      do 1 i = 1,4
           xx(i) = x(i,nxt)
    1 continue
c
c     Van der Pohl's eom
c
      f(1,nxt) = x(2,nxt)
      f(2,nxt) =-x(1,nxt)+(e-e*x(1,nxt)*x(1,nxt))*x(2,nxt)
c
c     A(t) matrix
c
      call amat(nxt,a)
c
c     delta_x dot = a delta_x
c
c
      do 50 i = 1,2
         f(i+2,nxt) = 0.d0
              do 50 k = 1,2
                   f(i+2,nxt) = f(i+2,nxt) + a(i,k)*x(k+2,nxt)
   50 continue
c
c
c
      return
      end
```

```
c
c
      subroutine amat(nxt,a)
c
c     calculate A matrix for Van der Pol's 'simple ' system.
c
      common /ham/ t,x(10,4),f(10,4),err(10),nn,hh,mode,e
      double precision t,x,f,err,hh,e
      double precision xx(2),a(2,2)
c
c
c     extract state
c
      do 5 i = 1,2
           xx(i) = x(i,nxt)
    5 continue
c
c
       a(1,1)=0.d0
       a(1,2)=1.d0
       a(2,1)=-1.d0-2.d0*e*xx(1)*xx(2)
       a(2,2)=e-e*xx(1)*xx(1)
c
      return
      end
```

```
c
c
      program pwr
c
c     do power spectral density calcs on deltax data
c
      dimension data(20000),psd(10000)
c
c     read in data file
c
      read (*,*) npts,tmax
      n = npts/2
      do 100 i = 1,npts
            read (*,*) t,data(i)
  100 continue
c
      call power( data, n, psd, tmax, fmax )
c
c     output
c
      open(1,FILE='psd.out',STATUS='UNKNOWN')
      df = fmax/real(n)
c
      do 200 i = 1,n+1
            f = real(i-1)*df
            write (1,1) f,psd(i)
    1       format(1x,e12.5, 1x, e12.5)
  200 continue
c
      stop
      end

c$INCLUDE: 'power.for'
c$INCLUDE: 'realft.for'
c$INCLUDE: 'four1.for'
      include 'power.for'
      include 'realft.for'
      include 'four1.for'
```

```
      subroutine power(data, n, psd, tmax, fmax)
c
c     one sided power spectral density per unit time of real function
c     (2n real values) stored in array data.  Performs fft
c     on real function using realft and four1, then returns
c     power spectral density in psd, n values.  For power spectral
c     density PER UNIT TIME, its divided by tmax, the length
c     of the time interval.  Also returns fmax, maximum frequency.
c     n MUST be a power of 2!  Returns n+1 values of psd, starting
c     at frequency 0 and going to fmax
c     output from realft is in cycles/sec, converted to
c     RADIANS/SEC!, psd in power per radian per unit time!
c
      dimension data(2), psd(2)
c
c     fft of real function
c
      call realft( data, n, +1)
c
c     conversion to per radian per unit time
c
      div = 1./( 2.*3.14159*tmax )
c
c     extract first and last (real valued) transformed points
c
      psd(1) = data(1)*data(1)*div
      psd(n+1) = data(2)*data(2)*div
c
c     remaining values are complex, take their norm
c
      do 100 i = 2,n
c
           i2m1 = 2*i-1
           i2 = 2*i
           psd(i) = ( data(i2m1)*data(i2m1) + data(i2)*data(i2) )
     1               *div
c
  100 continue
c
      dt = tmax / (2.* real(n) )
      fmax = 2.*3.14159/ (2.*dt)
c
      return
      end
```

```
c
c
      subroutine realft( data, n, isign )
c
c     calculates the fourier transform of a set of 2n real valued
c     data points.  Transliterated from c version of numerical
c     recipies.
c     The data stored in data(1.....2n) are replaced by the positive
c     frequency half of the complex fourier transform.  The real
c     valued first and last points are returned in data(1) and
c     data(2).  n MUST be a power of 2!  Also calculates the inverse
c     transform of a complex array if it is the transform of real
c     data, but the result must be divided by n.
c
c     isign = +1 for time -> frequency domain
c           = -1 for frequency -> time domain
c
      dimension data(2)
c     double precision for recurrences
      double precision wr,wi,wpr,wpi,wtemp,theta
c
      c1 = 0.5
c
c     initialize recurrences
c
      theta = 3.141592653589793d0/dble(n)
c
      if(isign .eq. 1) then
c           forward transform
            c2 = -0.5
c           do forward transform
            call four1(data, n, +1)
      else
c           set up for an inverse transform
            c2 = 0.5
            theta = -theta
      endif
c
c     prepare to separate two separate transforms of alternating
c     real data, and recombine into one transform on positive freq.
c
      wtemp = dsin( 0.5d0*theta )
      wpr = -2.d0*wtemp*wtemp
      wpi = dsin(theta)
      wr = 1.d0 + wpr
```

```
      wi = wpi
      n2p3 = 2*n + 3
c
c     deconvolution loop, case i=1 done separately below
c
      no2 = n/2
      do 100 i = 2,no2
           i1 = i + i - 1
           i2 = 1 + i1
           i3 = n2p3 - i2
           i4 = 1 + i3
c          the two separate transforms are separated out of data
           h1r =  c1*( data(i1) + data(i3) )
           h1i =  c1*( data(i2) - data(i4) )
           h2r = -c2*( data(i2) + data(i4) )
           h2i =  c2*( data(i1) - data(i3) )
c          then recombined to form the transform of the original data
           data(i1) =  h1r + wr*h2r - wi*h2i
           data(i2) =  h1i + wr*h2i + wi*h2r
           data(i3) =  h1r - wr*h2r + wi*h2i
           data(i4) = -h1i + wr*h2i + wi*h2r
c          trig recurrence
           wtemp = wr
           wr = wtemp*wpr - wi*wpi + wr
           wi = wi*wpr + wtemp*wpi + wi
  100 continue
c
c     final fix on forward transform, or inverse transform
c
      if(isign .eq. 1) then
c          squeeze first and last pts into first data slot
           h1r = data(1)
           data(1) = h1r + data(2)
           data(2) = h1r - data(2)
      else
c          do the inverse transform
           h1r = data(1)
           data(1) = c1*( h1r + data(2) )
           data(2) = c1*( h1r - data(2) )
           call four1(data, n, -1)
      endif
c
      return
      end
```

```
c
c
      SUBROUTINE FOUR1(DATA,NN,ISIGN)
c
c     fast fourier transform for complex single precision data
c     from numerical recipies.
c     data:  input real vector of complex valued function,
c            stored in time ascending order, with real/imag
c            parts alternating.
c     nn:    the number of complex data points.  data will actually
c            have 2 nn entries
c            IT IS VERY DESIRABLE THAT nn BE A POWER OF 2!
c     isign: direction of transform.  If isign = +1, goes from
c            time -> frequency domain.  If isign = -1, from
c            frequency -> time domani EXCEPT output transform needs
c            to be divided by nn to be normalized.
c
c     double precision for trig recurrences only
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION DATA(*)
c
c     bit reversal section of routine
c
      N=2*NN
      J=1
      DO 11 I=1,N,2
           IF(J.GT.I)THEN
c               exchange the two complex numbers
                TEMPR=DATA(J)
                TEMPI=DATA(J+1)
                DATA(J)=DATA(I)
                DATA(J+1)=DATA(I+1)
                DATA(I)=TEMPR
                DATA(I+1)=TEMPI
           ENDIF
           M=N/2
    1      IF ((M.GE.2).AND.(J.GT.M)) THEN
                J=J-M
                M=M/2
                GO TO 1
           ENDIF
           J=J+M
   11 CONTINUE
c
c     here begins the Danielson-Lanczos section of the routine
```

```
c     recurrent FFT's on power of 2 data subsets
      MMAX=2
c
c     outer loop executed log 2 nn times
c
2     IF (N.GT.MMAX) THEN
           ISTEP=2*MMAX
c          initialize trig recurrences
           THETA=6.28318530717959D0/(ISIGN*MMAX)
           WPR=-2.D0*DSIN(0.5D0*THETA)**2
           WPI=DSIN(THETA)
           WR=1.D0
           WI=0.D0
c
c          two nested inner loops
c
           DO 13 M=1,MMAX,2
                DO 12 I=M,N,ISTEP
                    J=I+MMAX
c                   the Danielson-Lanczos formula
                    TEMPR=SNGL(WR)*DATA(J)-SNGL(WI)*DATA(J+1)
                    TEMPI=SNGL(WR)*DATA(J+1)+SNGL(WI)*DATA(J)
                    DATA(J)=DATA(I)-TEMPR
                    DATA(J+1)=DATA(I+1)-TEMPI
                    DATA(I)=DATA(I)+TEMPR
                    DATA(I+1)=DATA(I+1)+TEMPI
   12               CONTINUE
c                   trig recurrences
                    WTEMP=WR
                    WR=WR*WPR-WI*WPI+WR
                    WI=WI*WPR+WTEMP*WPI+WI
   13          CONTINUE
               MMAX=ISTEP
c              end outer loop
               GO TO 2
          ENDIF
c
          RETURN
          END
```

Bibliography

1. Apostol, T. M. *Calculus Second Edition*. Addison-Wesley, 1957.
2. Floquet, M. G. *Ann. Ecole Normale Sup., Ser.2,12*, 47 (1883).
3. G. Benettin, L. Galgani, A. Giorgilli and J. Strelcyn *Meccanica, Volume 15*, 9 (1980).
4. Grossman, Stanley I. *Calculus Second Edition*. New York: Academic Press, Inc., 1981.
5. Jordan, D. W. and P. Smith. *Nonlinear Ordinary Differential Equations (Second Edition)*. New York: Oxford University Press, 1987.
6. Lyapunov, A. M. *Ann. Fac. Sci. Univ., Volume 9*, 203 (1947).
7. Moon, Francis C. *Chaotic Vibrations An Introduction for Applied Scientists and Engineers*. New York: John Wiley and Sons, Inc., 1987.
8. Reid, J. Gary. *Linear System Fundamentals Continuous and Discrete, Classic and Modern*. New York: McGraw-Hill, Inc., 1983.
9. Shimada, I. and T. Nagashima *Progress of Theoretical Physics, Volume 61*, 1605 (1979).
10. Strang, Gilbert. *Linear Algebra and its Applications*. San Diego: Harcourt Brace Jovanovich, 1957.
11. Wiesel, Dr. William E. "Extended Lyapunov Exponents," *Physical Review A* (In Press).
12. William T. Vetterling, [et.al.]. *Numerical Recipes (FORTRAN)*. New York: Cambridge University Press, 1985.

Vita

Captain Janice M. Horn was born on 22 November 1953 in Evergreen Park, Illinois. She entered the Air Force in 1977, and attended Auburn University at Montgomery, San Antonio College, and the University of Texas at Austin. Captain Horn earned her Bachelor of Science Degree in Aerospace Engineering from the University of Texas in 1987 under the Airman's Education and Commissioning Program. Following Officer Training School, she was assigned to the 6555 Aerospace Test Group at Cape Canaveral Air Force Station, Florida. While assigned to the Test Group, she worked spacecraft integration on expendable launch vehicles, Atlas II activation, and was the Environmental Engineering Project Officer for the Test Group. In 1991, Captain Horn began working in the area of sensor development/data analysis for the Air Force Technical Applications Center. She entered the Air Force Institute of Technology, School of Engineering in May, 1991.

Permanent address: 233 Chatham Drive
Fairborn, Ohio 45324

15 December 1992 Master's Thesis

EXTENDED LYAPUNOV EXPONENTS FOR SECOND ORDER SYSTEMS

Janice M. Horn
Captain, USAF

Air Force Institute of Technology
WPAFB OH 45433-6583

AFIT/GA/ENY/92D-09

NONE

It is shown that the concept of the Lyapunov exponent can be extended to include an *imaginary* part. A numerical technique used to calculate these *extended* Lyapunov exponents for second order systems is presented. There are two requirements for this extension: the definition of a coordinate frame on the tangent space of the differential equation, and an extension of the classical limit, called the *limit of the mean*. An application of the technique to the van der Pol equation for the constant coefficient and periodic coefficient cases is given. The extended Lyapunov exponents found using this technique totally agree with the eigenvalues for the constant coefficient case. In the periodic coefficient case, not only do the extended Lyapunov exponents agree with the Poincaré exponents calculated using standard Floquet theory, they confirm that the imaginary parts of the Poincaré exponents are equal to the quotient $\pm i\frac{(2\pi)}{\tau}$ where τ is the period of the trajectory. This imaginary part is uncertain when using standard Floquet theory. Additionally, fast Fourier transform (FFT) techniques are used to validate the existence of the extended Lyapunov exponent and the values obtained for its imaginary part. These techniques show that the power spectrum of relative motion is discrete for the trial cases presented, with the fundamental frequency almost exactly equal to the calculated imaginary part of the extended Lyapunov exponent. Coupled with the successful comparison of characteristic exponents for the constant coefficient and periodic coefficient cases, this power spectrum serves to decisively validate the existence of the *extended* Lyapunov exponent.

Lyapunov Exponents, Stability, Fundamental Frequency, Eigenvalues, Poincaré Exponents 116

Unclassified Unclassified Unclassified UL