Applications of Classical Non-Linear Liouville Dynamic Approximations

by

Terry Lee Harter

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in

Physics

APPROVED:

Samuel P. Bowen, Chairman

L. D. Roper

C. D. Williams

R. L. Bowden

R. K. P. Zia

December, 1988

Blacksburg, Virginia

Applications of Classical Non-Linear Liouville Dynamic Approximations

by

Terry Lee Harter

Committee Chairman: Samuel P. Bowen
Department of Physics

(ABSTRACT)

This dissertation examines the application of the Liouville operator to problems in classical mechanics. An approximation scheme or methodology is sought that would allow the calculation of the position and momentum of an object at a specified later time, given the initial values of the object's position and momentum at some specified earlier time. The approximation scheme utilizes matrix techniques to represent the Liouville operator.

An approximation scheme using the Liouville operator is formulated and applied to several simple one-dimensional physical problems, whose solution is obtainable in terms of known analytic functions. The scheme is shown to be extendable relative to cross products and powers of the variables involved. The approximation scheme is applied to a more complicated one-dimensional problem, a quartic perturbed simple harmonic oscillator, whose solution is not capable of being expressed in terms of simple analytic functions. Data produced by the application of the approximation scheme to the perturbed quartic harmonic

oscillator is analyzed statistically and graphically. The scheme is reapplied to the solution of the same problem with the incorporation of a drag term, and the results analyzed. The scheme is then applied to a simple physical pendulum having a functionalized potential in order to ascertain the limits of the approximation technique. The approximation scheme is next applied to a two-dimensional non-perturbed Kepler problem. The data produced is analyzed statistically and graphically. Conclusions are drawn and suggestions are made in order to continue the research in several of the areas presented.

ACKNOWLDEGEMENTS

Not wishing to become intellectually stagnant, I began pursuing graduate studies almost immediately after my arrival at, what was then, the Naval Weapons Laboratory by enrolling in the off-campus program operated by Virginia Tech, who, shortly thereafter, had just hired a unique individual to teach, administer, and run the off-campus program in Physics. That individual was Dr. Samuel P. Bowen. Due to a work committment in California, my graduate program for the masters degree was interrupted, and I began to make trips to Blacksburg to meet with Dr. Bowen to alleviate incompletes. After obtaining a master's degree, studies were continued in a off-campus situation. When the opportunity to become a full-time graduate student in residence presented itself, I applied for it.

It is with a great deal of gratitude and appreciation that I now take the opportunity to acknowledge and hereby thank the many people that have provided me assistance in one form or another over the years while I was pursuing graduate study. Foremost among these is my advisor and friend, Dr. Samuel P. Bowen. He suggested a dissertation topic that he thought would eventually be useful to the Navy, a requirement for obtaining support for the graduate research program. He has been an unending source of

inspiration to me during these endeavors. When circumstances looked bleak, to me at least, he has always been optimistic and encouraging. I will always value the many brief conversations that we have had together discussing ideas, laying out the best approach to solve a critical problem, and in the oral reporting of progress on the work of this dissertation. It is to his credit that, when considering all other academic disciplines offered at NSWC, there have been more graduates in physics at NSWC at both the Masters and Ph.D. levels than any other discipline. That track record is a tribute to his success.

I would also like to thank the members of my dissertation committee, most of whom I have had the oppportunity to take a class from.

Being a full time civilian employee of the Department of the Navy at the Naval Surface Warfare Center has provided both advantages and disadvantages during the period. The obvious advantages include financial support provided by the opportunity to participate in the Full Time Study Program during 1984-1985, allowing me to take up full-time residence at the university for that timeframe, and by the opportunity to partcipate in the Graduate Research Program during the 1987-1988 academic year, again allowing me the chance to take up residency at the university. The uninterrupted period of time away from the

interruptions of work will always be remembered and cherished. The disadvantage has been the time constraint imposed (even if artificial and self imposed in nature), feeling that one must be finished by a certain time, and not allowing for the investigation of interesting but tangential topics unearthed by the progress of the research. There are several individuals at NSWC that I would like to thank and acknowledge. Foremost among these is Dr. Charles F. Fennemore, my supervisor and colleague, who, I suspect, has looked after my interests and run interference on my behalf more times than I probably know. His support is greatfully acknowledged. I would also like to thank Dr. Normand Auger, a past branch head who initially supported me for the Full Time Study Program, and , a past and current branch head, who approved me for both the Full Time Study Program and, later, the Graduate Research Program. Thanks is also extended to , who, as my division head, approved me for the Full Time Study Program, and who later, as assistant department head, approved me for the Graduate Research Program.

Among the other people that I am greatly indebted to is my brother, , who provided me with the necessary communicational support, via forewarding important mail and in relaying information to and from my

employer, and with the problems associated with maintaining two residences while I was in residence in Blacksburg.

Among the fellow students that I have had the good fortune to be associated with, the one that I have had the longest acquaintance with is , also a fellow NSWC employee, who provided support and encouragement during times of stress, and showed me that it was, indeed, possible to obtain a Ph.D. while a full time civilian employee of the Navy. Among the graduate students on campus, I must single out and pay special thanks to , who helped me with the commands of the RIDGE minicomputer. The friendship of , , , , , and is also acknowledged and appreciated.

Finally, one individual deserves very special thanks for assisting me in getting the right forms to the right people at the right time, always willing to help in executing the required transitional paperwork from off-campus to on-campus student status and back again, and back again is . Her assistance is greatly appreciated and acknowledged.

Table of Contents

List of Tables

List of Figures

FIGURE CAPTIONS

CHAPTER 1

INTRODUCTION AND THEORY

This dissertation describes an attempt to find a new method that embodies an approximation scheme allowing the Liouville operator to be expressed via a matrix represention in a practical, easily realizable implementation which can be formulated into a calculational scheme. The Liouville operator is defined, in the classical case as the Poisson bracket of the function to be operated upon and the Hamiltonian, H. The Liouville operator in the quantum context has been used in other methods including a group of convergent methods known as Liouville resolvent methods (Bowen 1975) which are designed to construct a sequence of approximants useful in calculating approximate thermodynamic Green's functions for some specified Hamiltonians. The Liouville resolvent method has produced exact solutions when simple Hamiltonians are considered (Bowen et al 1984). This dissertation will be limited to the study of the Liouville operator in areas of classical mechanics where the Hamiltonian is time independent.

A study of different methodologies was performed to identify plausible approaches capable of yielding useful

results. Consideration will be given to a method utilizing the best representation and a reasonable rate of convergence that can be quantified by a measure of the errors involved when implemented under different situations. One goal of the research is the computational improvement, in speed and accuracy of the determination of position and velocity of a satellite in a perturbed Kepler problem, for example. Virtually all currently implemented solutions of such a problem now employ Cowell's method (Bate et al 1971). Cowell's method is a technique that obtains an iterative numerical integration solution whose time range validity is [usually] relatively small for the computational effort required to produce the solution. Our search for an approximation scheme will be guided by the objective of trying to keep the computational effort to a minimum. Application of the matrix technique will then yield analytical expressions of approximinants capable of enhancing computational algorithms for the short time regime. The above provides much of our motivation in pursuing this research.

A short discussion of the problem to be investigated and the proposed method and techniques used in the main body of the text is presented next. Let $\mathcal{L}$ denote the Liouville operator, defined by

$$\mathcal{L}(\tau) = [\tau, H], \qquad (1.1)$$

where τ is some arbitary function of interest, where [,] is the classical Poisson Bracket, and H is the Hamiltonian of the system. For any function u(t) defined on phase space $\Gamma(x,p)$, it is well known that (for $\vec{u} = (x,p)^T$)

$$\frac{d\vec{u}}{dt} = \left[\vec{u}, H\right] = \mathcal{L}(\vec{u}), \qquad (1.2)$$

which has the formal solution

$$\vec{u}(t) = e^{\mathcal{L}t}\vec{u}\Big|_0 . \qquad (1.3)$$

Specializing to the variables x and p, the solutions as a function of time are

$$x(t) = e^{\mathcal{L}t}x\Big|_o = \sum_{n=0}^{\infty}\frac{t^n}{n!}\left\{\mathcal{L}^n \quad x\right\}\Big|_o \qquad (1.4a)$$

and

$$p(t) = e^{\mathcal{L}t}p\Big|_o = \sum_{n=0}^{\infty}\frac{t^n}{n!}\left\{\mathcal{L}^n \quad p\right\}\Big|_o . \qquad (1.4b)$$

The Laplace transform of these equations gives the frequency dependence of x and p, as

$$\hat{x}(s) = \int_o^{\infty} e^{-st}e^{\mathcal{L}t}x(0)\,dt = (s-\mathcal{L})^{-1}x(0) = \sum_{n=0}^{\infty}\frac{(\mathcal{L}^n x)\,|_o}{s^{n+1}} \qquad (1.5a)$$

and

$$\hat{p}(s) = \int_0^\infty e^{-st} e^{\mathcal{L}t} p(0)\,dt = (s-\mathcal{L})^{-1} p(0) = \sum_{n=0}^{\infty} \frac{(\mathcal{L}^n p)|_0}{s^{n+1}}, \tag{1.5b}$$

where the (hat) notation, $\hat{x}(s)$, is utilized to indicate that the time variable has been Laplace transformed. Examination of the right side of the equations (1.5) shows that they are moment expansions, hence the calculation of all of the required terms can be performed provided a truncation limit exists for some suitable integer N. The central question is how to represent the inverse operator $(s\mathbf{I}-\mathcal{L})^{-1}$ or resolvent using matrix techniques. If such a representation is found, the time dependence can be recovered by taking the inverse Laplace transform. Note that the resultant expression is dependent upon the initial conditions.

Suppose, for the moment, that we have found a way to relate the operator $\mathcal{L}$ to its matrix analog, denoted by L. The eigenvalues of the matrix L are denoted by λ_i, and the eigenvectors are denoted by $\vec{\xi}_i$, where $1 \leq i \leq N$. In general, both the eigenvalues and components of the eigenvectors $\vec{\xi}_i$ can be complex. Let $\hat{\Phi}_0$ denote the vector in the transformed space containing the initial condition information, with

$$\hat{\Phi}_0 = \begin{bmatrix} \hat{x}(0) \\ \hat{p}(0) \\ \hat{\Phi}_3(0) \\ \hat{\Phi}_4(0) \\ \vdots \\ \hat{\Phi}_N(0) \end{bmatrix}, \qquad (1.6)$$

where $\hat{x}$ and $\hat{p}$ are taken as the first and second components of $\hat{\Phi}_0$. The first two components of Φ_0 are chosen as $\hat{x}$ and $\hat{p}$ since the position and momentum are the observables we are most interested in. The components $\hat{\Phi}_3$ through $\hat{\Phi}_N$ must be chosen as linearly independent of each other and of $\hat{x}$ and $\hat{p}$. Since the $\hat{\Phi}_i$ are linearly independent, the set of $\vec{\xi}_i$ (the eigenvectors) are also independent, and one should be able to solve the system of equations

$$\begin{bmatrix} \hat{x}(0) \\ \hat{p}(0) \\ \hat{\Phi}_3(0) \\ \hat{\Phi}_4(0) \\ \hat{\Phi}_5(0) \\ \vdots \\ \hat{\Phi}_N(0) \end{bmatrix} = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ [\vec{\xi}_1], & [\vec{\xi}_2], & \ldots\ldots, & [\vec{\xi}_N] & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_N \end{bmatrix}, \qquad (1.7)$$

for the α_i coefficients if the basis eigenvectors $\vec{\xi}_i$ are known. In the above matrix equation $[\vec{\xi}_i]$ denotes the column of components of the i^{th} eigenvector. Then since

$$\hat{\Phi}_o = \sum_{i=1}^{N} \alpha_i [\vec{\xi}_i],$$

and $\mathcal{L}\vec{\xi}_i \triangleq L\vec{\xi}_i = \lambda_i \vec{\xi}_i$, we have

$$\hat{\Phi}(s) = (s-\mathcal{L})^{-1}\hat{\Phi}_o = (s-\mathcal{L})^{-1}\left\{\sum \alpha_i [\vec{\xi}_i]\right\} = \sum \alpha_i \left\{(s-\mathcal{L})^{-1}\vec{\xi}_i\right\}$$

$$\hat{\Phi}(s) \triangleq \sum \frac{\alpha_i \vec{\xi}_i}{(s - \lambda_i)}, \tag{1.8}$$

which can then be inverted to obtain the time dependent vector $\vec{\Phi}(t)$, whose first two components are $x(t)$ and $p(t)$. Taking the inverse transform of the above, we obtain

$$\vec{\Phi}(t) = \frac{1}{2\pi i}\oint e^{st}\hat{\Phi}(s)\,ds \triangleq \sum_j \alpha_j [\vec{\xi}_j] \oint \frac{e^{st}}{2\pi i\,(s - \lambda_j)}\,ds$$

$$\vec{\Phi}(t) \triangleq \sum_j \alpha_j [\vec{\xi}_j]\, e^{\lambda_j t}, \tag{1.9}$$

where the notation $[\vec{\xi}_j]$ denotes the j^{th} column eigenvector of L. The above outline constitutes an approach that could be used. This Laplace transform approach requires a knowledge of both the eigenvalues and their associated eigenvectors. Generally, eigenvalues are usually easier to obtain than eigenvectors. For some matrices eigenvectors are not easily obtained due to numerical instabilities in

the components which present difficulties in the determination of eigenvectors.

Consequently, a method that avoids eigenvector determination will be sought. We return to the expression for the formal solution given in equation 1.3 . The critical term in this formal solution is the exponentiation of the operator involved. At the heart of the approximation is the replacement of the operator in the exponential

$$e^{\mathcal{L}t} \cong e^{Lt}, \qquad (1.10)$$

by an n×n matrix L. This suggests that the matrix exponential be examined. We focus upon finding a technique relating $\mathcal{L}$ to its equivalent in matrix form, L. No mention has been made on how large a dimension L should have in order to have an effective representation of $\mathcal{L}$.

The basis set chosen to represent the matrix L determines the form of the elements of the matrix. One of the problems is determining criteria with which to pick a basis set. Once this basis set has been selected, the initial mode of attack will be to apply the Liouville operator to the initial basis, and then examine the resultant expressions produced. Do the results contain linear combinations of the initial basis? Are any terms in the resultant expression not expressible as a linear

combination of the initial basis vectors? If so, can these new terms be categorized or classified in any way?

The choice of bases that will be tried initially will be simple monomials and polynomials of position and momentum, taking a clue from the work of Douglas (Douglas 1982). Two possibilities for the dimensionality of L come immediately to mind. The first possibility is that for which the dimension of L is finite. This case is frequently encountered when the problem has an exact solution that is easily obtained. The other possibility is one in which the dimensionality of L is infinite. The expectation is that the individual nature of the Hamiltonian determines which of the two possiblities is realized.

The justification that our approximation scheme will work has its roots in the work of Masson (Masson 1970:197). Working on a quantum mechanical problem, Masson has shown that, once an orthogonal basis set of vectors is picked through which the resolvent $(s - \mathcal{L})^{-1}$ is represented, increasing the size of the basis set used by passing to larger truncations of the matrix should increase the accuracy of the approximation provided by the resolvent. Although the resolvent techniques require that the operators involved be self-adjoint, a more general approach can be used that does not require that the

operator be self-adjoint. This approach is termed the method of moments (Masson 1970:206). While the method of moments is not used explicitly here, an equivalent method employing the matrix exponential is used. The approximation technique of building a sequence of larger and larger matrices is applied hoping to find similar improvement in the specific problems which we are considering. An inner product has not been found for the space in which we are working, consequently the space does not qualify as a Hilbert space. The space may be a Banach space if the absolute value function is taken as the norm of the space. The sets of polynomials and monomials with which we will be working with are known to be complete in the Banach spaces of continuous functions (Riesz & Nagy 1955). A set of theorem(s) similar to Masson's theorem(s) requiring Hilbert spaces needs to be proven for Banach spaces. We will not provide proofs of the necessary theorem(s) here. No functional analysis has been conducted on the spaces involved, because this was not deemed the intent of the research, however such an analysis may yet have to be performed. As will be seen, the approximation scheme produced good results for a short time then the results diverged from the exact solution. A concise analysis of the possible causes of this behavior is presented in the last chapter. Current thinking is that

not all of the prerequistes of Masson's theorems were met. Either the approximation scheme will have to be repeated using the bases and operators meeting the Masson theorems or new theorems analogous to the Masson theorems will have to be proven.

Our plan of attack will now be briefly outlined. A method was found that is only dependent upon the eigenvalues, regardless of their multiplicities. This procedure calculates a matrix exponential by using the Putzer algorithm. The approximation scheme is formulated using this algorithm and in Chapter 2 several examples applying this approximation plan to simple Hamiltonians will be presented. A more complicated test of the approximation technique will be conducted in Chapter 3. This chapter examines the performance of the approximation methodology when applied to a perturbed simple harmonic oscillator. An examination of how the approximation methodology performs when a dissipative (drag) force is introduced to the same problem will be conducted in Chapter 4. Exploration of how the approximation scheme behaves when monomials and polynomials are omitted and a simple function of position is employed in the Hamiltonian is the objective of Chapter 5. The approximation method is applied to the unperturbed Kepler problem in Chapter 6 to ascertain how the approximation scheme handles a two-

dimensional problem. Finally, Chapter 7 offers conclusions and ideas, areas, and suggestions by which the research might be continued and extended. A unifying discussion of the behavior of the approximation scheme as applied to the test of the various classical mechanical problems contained in this dissertation is also presented.

CHAPTER 2

PUTZER THEOREM, SIMPLE APPLICATIONS TO CLASSICAL PROBLEMS

In this chapter, an examination of an evaluation procedure for the matrix operator e^{At} is conducted. Standard evaluation techniques do not generally work, so we use the Putzer algorithm. After exhibiting how to apply this algorithm, the application of the approximation scheme incorporating this algorithm to several simple Hamiltonians each having an exact analytical solution is conducted. This will serve as a standard by which to examine Hamiltonians that do not have exact solutions.

Having seen the exponential matrix e^{At} as the key to the approximation procedure, a practical evaluation scheme is sought that separates out the time dependence and allows that time dependence to be evaluated analytically. This process must also allow for the proper treatment of degenerate eigenvalues (those eigenvalues having multiplicities greater than one) of the matrix A. Attention is now focused on a viable means of constructing and calculating the matrix exponential. The standard series definition is not suitable due to the infinite sum involved. A good survey of various techniques is the one performed by Moler and Van Loan (Moler and Van Loan 1978).

A practical and straight-forward technique was found that permits the calculation of e^{At}, where A is an n×n matrix of real constants. This technique, based upon the Cayley-Hamilton Theorem and known as Putzer's algorithm of calculating the matrix exponential, is dependent only upon the eigenvalues of the matrix, A, and takes into account the multiplicity associated with each eigenvalue of A. (Putzer 1966):

Putzer's Theorem:

Let A be an n×n matrix with possibly non-distinct eigenvalues $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \ldots, \lambda_n$, listed in some arbitrary but specified order. Then an explicit formula for e^{At} is given by

$$e^{At} = \sum_{j=0}^{n-1} r_{j+1}(t)\, P_j \quad , \qquad (2.1)$$

where, if **I** represents the identity matrix,

$$P_0 = \mathbf{I},$$

$$P_j = \prod_{k=1}^{j} (A - \lambda_k \mathbf{I}) \qquad j = 1, 2, \ldots, n-1 \qquad (2.2)$$

and the functions $r_1(t), \ldots, r_n(t)$ are solutions of the triangular system of equations expressible in matrix form as

$$\begin{bmatrix} \\ \dfrac{d\vec{r}}{dt} \\ \\ \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \lambda_2 & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 1 & \lambda_3 & 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 0 & 1 & \lambda_4 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & 1 & \cdot & \lambda_{n-2} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \lambda_{n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \lambda_n \end{bmatrix} \begin{bmatrix} \\ \vec{r} \\ \\ \end{bmatrix}, \tag{2.3}$$

with the initial conditions that

$$r_1(t = 0) = 1$$
$$r_j(t = 0) = 0 \quad \text{for} \quad j = 2, \ldots, n.$$

Note that for eigenvalues having multiplicity greater than one, the frequency of appearance of these eigenvalues along the diagonal of the above matrix is in accordance with their multiplicity. In general, the n^{th} function, r_n, is given in terms of the previous function r_{n-1} as

$$r_n(t) = e^{\lambda_n t} \int_0^t e^{-\lambda_n \xi} r_{n-1}(\xi)\, d\xi \qquad n \geq 2 \tag{2.4}$$

with the initial value condition $r_n(0) = 0$. For higher dimensional L matrices calculation of the $r_k(t)$ functions becomes gradually more difficult and prone to error. An alternate means of calculating the scalar $r_k(t)$ functions is now presented.

In general, the most effective way to solve for the

scalar functions involves taking the Laplace transform of the defining differential equation of the Putzer algorithm and utilizing the proper initial conditions, solving algebraically for the function and taking the inverse transform. The differential equation of the Putzer algorithm is

$$\dot{r}_j(t) = r_{j-1}(t) + \lambda_j r_j(t)$$

$$\text{with} \quad r_1(0) = 1 \quad \text{and} \quad r_j(0) = 0 \quad j \geq 2 .$$

Taking the Laplace transform of the differential equation one obtains

$$s\,\hat{r}_j - r_j(0) = \hat{r}_{j-1} + \lambda_j \hat{r}_j$$

or

$$\hat{r}_j = \frac{\hat{r}_{j-1} + r_j(0)}{(s - \lambda_j)} .$$

Two cases exist, for $j = 1$, and for $j \geq 2$. For $j = 1$, the above equation reduces to

$$\hat{r}_1 = \frac{1}{s - \lambda_1} .$$

since $\hat{r}_0 = 0$ by definition. For $j \geq 2$ the above equation yields

$$\hat{r}_j = \frac{\hat{r}_{j-1}}{s - \lambda_j} = \frac{1}{\prod_{b=1}^{j} (s - \lambda_b)} ,$$

since $r_j(0) = 0$. One could then take the inverse Laplace transform to obtain the time dependent function once the

eigenvalue spectrum is known. Under either method, $r_i(t)$ is given by

$$r_i(t) = e^{\lambda_i t} .$$

As examples of the application of the Putzer algorithm, the approximation technique will be applied to several simple physical systems. The first illustration will be that of the unperturbed simple harmonic oscillator. The Hamiltonian is

$$H = \frac{p^2}{2m} + \frac{1}{2} k x^2 .$$

Application of $\mathcal{L}$ on the initial basis, x and p, gives

$$\mathcal{L}(x) = p/m ,$$

$$\mathcal{L}(p) = -k x ,$$

so that the corresponding L matrix is

$$L = \begin{bmatrix} 0 & \frac{1}{m} \\ -k & 0 \end{bmatrix} .$$

The eigenvalues are $\pm i\sqrt{k/m}$, denoted by $\pm i\omega$. The $r(t)$ functions are

$$r_i(t) = e^{i\omega t} ,$$

$$r_2(t) = \frac{1}{\omega} \sin(\omega t) ,$$

while the P matrices are $P_0 = \mathbf{I}$, and

$$P_1 = \begin{bmatrix} -i\omega & \frac{1}{m} \\ -k & -i\omega \end{bmatrix} .$$

The matrix for e^{Lt} is

$$e^{Lt} = \begin{bmatrix} \cos(\omega t) & \frac{1}{m\omega} \sin(\omega t) \\ -m\omega \sin(\omega t) & \cos(\omega t) \end{bmatrix} .$$

Employing $\vec{u}(t) = e^{Lt} \vec{u}(0)$, gives

$$x(t) = x_0 \cos(\omega t) + \frac{p_0}{m\omega} \sin(\omega t) ,$$

$$p(t) = -x_0 m\omega \sin(\omega t) + p_0 \cos(\omega t) ,$$

as required.

The second illustration will be that of the free particle. The Hamiltonian is

$$H = \frac{p^2}{2m} .$$

Taking x and p as the variables, the Liouville operator is applied, and the matrix

$$L = \begin{bmatrix} 0 & \frac{1}{m} \\ 0 & 0 \end{bmatrix} ,$$

becomes the representation of $\mathcal{L}$. The eigenvalues are both zero, producing

$$r_1(t) = 1 ,$$
$$r_2(t) = t ,$$

and $P_0 = \mathbf{I}$, $P_1 = L$. The Putzer algorithm then gives

$$e^{Lt} = \begin{bmatrix} 1 & \frac{1}{m} t \\ 0 & 1 \end{bmatrix} ,$$

yielding

$$x(t) = x_0 + \frac{p_0}{m} t ,$$
$$p(t) = p_0 ,$$

again as required.

The next application of the method will be to the one-dimensional constant-gravity Hamiltonian. (This application is examined to provide a framework in which to test a basis feature of the approximation method; see below.) The Hamiltonian is

$$H = \frac{p^2}{2m} + mgx ,$$

and the Liouville operator acting on x and p gives

$$\mathcal{L}(x) = [x, H] = \frac{p}{m} \quad , \quad \mathcal{L}(p) = [p, H] = -mg \quad .$$

If the vector $\vec{u}$ is taken as

$$\vec{u} = \begin{bmatrix} x \\ p \\ 1 \end{bmatrix} ,$$

where the constant 1 has been included into the initial basis so that the results of the Liouville operator upon p can be incorporated into the matrix producing

$$\frac{d\vec{u}}{dt} = \mathcal{L}\vec{u} = [\vec{u}, H] = \begin{bmatrix} 0 & 1/m & 0 \\ 0 & 0 & -mg \\ 0 & 0 & 0 \end{bmatrix} \vec{u} ,$$

and the characteristic equation is

$$\lambda^3 = 0 ,$$

so the roots are $\lambda_1=\lambda_2=\lambda_3=0$, yielding

$$r_1(t) = 1$$
$$r_2(t) = t$$
$$r_3(t) = \frac{1}{2} t^2$$

for the r functions, and $P_0 = \mathbf{I}$,

$$P_1 = \begin{bmatrix} 0 & 1/m & 0 \\ 0 & 0 & -mg \\ 0 & 0 & 0 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 0 & 0 & -g \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

for the P matrices. The Putzer theorem thus gives

$$e^{At} = \sum_{J=0}^{2} r_{J+1}(t)\, P_J = r_1(t)P_0 + r_2(t)P_1 + r_3(t)P_2$$

$$= \begin{bmatrix} 1 & t/m & -\frac{1}{2}gt^2 \\ 0 & 1 & -mgt \\ 0 & 0 & 1 \end{bmatrix}$$

Then $\vec{u}(t) = e^{Lt}\,\vec{u}_0$ gives

$$\begin{bmatrix} x(t) \\ p(t) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & t/m & -\frac{1}{2}gt^2 \\ 0 & 1 & mgt \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ p_0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 + \frac{p_0}{m}t - \frac{1}{2}gt^2 \\ p_0 - mgt \\ 1 \end{bmatrix},$$

which is the result expected.

That the technique works on a set of extended variables can be seen by keeping the last Hamiltonian and using the set of variables $x^2, p^2, xp, x, p,$ and 1 as the components of the $\vec{u}$ vector, e.g.

$$\vec{u} = \begin{bmatrix} x^2 \\ p^2 \\ xp \\ x \\ p \\ 1 \end{bmatrix} .$$

The action of the Liouville operator on each of the above components produces

$$\mathcal{L}(x) = \frac{p}{m}$$

$$\mathcal{L}(p) = -mg$$

$$\mathcal{L}(x^2) = 2x\frac{p}{m}$$

$$\mathcal{L}(p^2) = -pmg$$

$$\mathcal{L}(xp) = \frac{p^2}{m} - xmg$$

so that the matrix L representing $\mathcal{L}$ for this basis is

$$\frac{d\vec{u}}{dt} = \mathcal{L}\vec{u} = \left[\begin{array}{ccc|ccc} 0 & 0 & 2/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2mg & 0 \\ 0 & 1/m & 0 & -mg & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1/m & 0 \\ 0 & 0 & 0 & 0 & 0 & -mg \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \vec{u} = L\vec{u} .$$

matrix for original problem

Again the eigenvalues are all zero, as the characteristic equation is $\lambda^6 = 0$. The six r functions are thus again

given by

$$r_n(t) = \frac{t^{n-1}}{(n-1)!}, \quad n = 1, 2, \cdots, 6;$$

while the P matrices are

$$P_0 = I,$$

$$P_1 = (L - \lambda_1 I) P_0 = \begin{bmatrix} 0 & 0 & 2/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2mg & 0 \\ 0 & 1/m & 0 & -mg & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/m & 0 \\ 0 & 0 & 0 & 0 & 0 & -mg \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$P_2 = (L - \lambda_2 I)(L - \lambda_1 I) P_0 = P_1^2$$

$$= \begin{bmatrix} 0 & 2/m^2 & 0 & -2g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(mg)^2 \\ 0 & 0 & 0 & 0 & -3g & 0 \\ 0 & 0 & 0 & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$P_3 = (L - \lambda_3 I) P_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & -6g/m & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3mg^2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$P_4 = (L - \lambda_4 \mathbf{I}) P_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 6g^2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

while the last P matrix for this example is the null matrix

$$P_5 = (L - \lambda_5 \mathbf{I}) P_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \mathbf{Z}.$$

Application of the Putzer theorem yields

$$e^{Lt} = \sum_{j=0}^{5} r_{j+1}(t)\ P_j =$$

$$= P_0 + t P_1 + \frac{1}{2}t^2 P_2 + \frac{1}{6}t^3 P_3 + \frac{1}{24}t^4 P_4 + \frac{1}{120}t^5 P_5 ,$$

so one obtains

$$e^{Lt} = \begin{bmatrix} 1 & t^2/m^2 & 2t/m & -gt^2 & -(g/m)t^3 & (1/4)g^2t^4 \\ 0 & 1 & 0 & 0 & -2mgt & (mgt)^2 \\ 0 & t/m & 1 & -mgt & -(3/2)gt^2 & (1/2)mg^2t^3 \\ 0 & 0 & 0 & 1 & t/m & -(1/2)gt^2 \\ 0 & 0 & 0 & 0 & 1 & -mgt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Then $\vec{u}(t) = e^{Lt}\vec{u}(0)$ gives

$$\begin{bmatrix} x^2(t) \\ p^2(t) \\ x(t)\,p(t) \\ x(t) \\ p(t) \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & t^2/m^2 & 2t/m & -gt^2 & -(g/m)t^3 & (1/4)g^2t^4 \\ 0 & 1 & 0 & 0 & -2mgt & (mgt)^2 \\ 0 & t/m & 1 & -mgt & -(3/2)gt^2 & (1/2)mg^2t^3 \\ 0 & 0 & 0 & 1 & t/m & -(1/2)gt^2 \\ 0 & 0 & 0 & 0 & 1 & -mgt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^2(0) \\ p^2(0) \\ x(0)\,p(0) \\ x(0) \\ p(0) \\ 1 \end{bmatrix},$$

which, when multiplied out, can be seen to contain the squares of x(t), p(t), and the cross term x(t)p(t), as well as x(t) and p(t) found from the previous 3 by 3 problem. Hence, the approximation procedure using the Putzer algorithm is valid under an extendable set of polynomial basis components. Thus the approach seems to be consistent in this regard.

Recapping the work performed in this chapter, the Putzer algorithm for calculating the matrix exponential was introduced and shown to work on several simple physical systems. The algorithm is capable of decomposing the time dependence of the matrix exponential into various scalar functions that are applied toward the matrix exponential by

the direction of intermediate matrices dictating where the time dependence will be applied in producing the overall matrix function. All possible multiplicities of eigenvalues of the matrix are able to be handled in a natural way and without the need for their associated eigenvectors. An alternative way of defining the necessary scalar functions containing the time dependence was suggested. The method was shown to work on an extendable set of basis elements. The simple analytic examples shown in this chapter illustrate the structure of the Putzer algorithm and show how larger polynomial sets of functions give a consistent representation of the motion of simple physical systems. The success of the technique on these simple physical systems suggests that the technique can be applied to more complex problems. Accordingly, we will next attack problems for which there are no simple analytical solutions.

CHAPTER 3

STUDY OF APPROXIMATION TECHNIQUE USING PUTZER ALGORITHM ON SIMPLE HARMONIC OSCILLATOR WITH X^4 PERTURBATION

Now that the Putzer technique of constructing a closed form solution for exp(Lt) has been shown to be tractable for simple problems of small dimension, the technique is next applied to a problem that has the capability of leading to larger and larger matrices. To achieve this, the Hamiltonian for the simple harmonic oscillator is chosen and modified to include a gx^4 pertubation term in the potential, with g being a constant. Thompson and Stewart refer to this potential as a "stiffening spring" potential for positive g, and a "softening spring" potential for negative values of g (Thompson and Stewart 1986). This chapter is confined to the investigation into the stiffening spring potential since the majority of data produced was for this class of potential. The suite of computer programs written to generate the required data is capable of handling both positive and negative values of g. The Hamiltonian for the simple harmonic oscillator with a stiffening potential is here expressed as

$$H = \frac{p^2}{2m} + \frac{1}{2}kx^2 + \frac{g}{4!}x^4 \quad . \qquad (3.1)$$

Hamilton's equations for this Hamiltonian can of course be combined to produce the second order, non-linear

differential equation (Newton's second law)

$$\ddot{x} = -\frac{k}{m}x - \frac{g}{3!\ m}x^3,$$

whose solution can be obtained in the form t(x) by quadratures

$$t + C_2 = \int \frac{dx}{\left[C_1 - \frac{k}{m}x^2 - \frac{2g}{4!\ m}x^4\right]^{1/2}},$$

with the result producing an elliptical integral. Rather than solve numerically this elliptical integal and inverting, a fourth-order numerical Runge-Kutta-Gill procedure may be applied directly to the set of first-order linear differential equations produced by Hamilton's equations (White 1974). This last solution will be referred to as the "exact" solution.

Returning to the present approximation technique, the monomials x and p are picked as the initial basis and the action of the Liouville operator on this basis is given by

$$\mathcal{L}(x) = (1/m)\,p,$$

$$\mathcal{L}(p) = (-k)\,x + (-g/3!)\,x^3,$$

so a new monomial, x^3, has been generated. Applying the

Liouville operator to this new monomial and on subsequent new monomials generated, one obtains

$$\mathcal{L}(x^3) = \underline{(3/m)}\,x^2p,$$

$$\mathcal{L}(x^2p) = \underline{(-k)}\,x^3 + \underline{(2/m)}\,xp^2 + \underline{(-g/3!)}\,x^5$$

$$\mathcal{L}(xp^2) = \underline{(-2k)}\,x^2p + \underline{(1/m)}\,p^3 + \underline{(-g/3)}\,x^4p$$

$$\mathcal{L}(p^3) = \underline{(-3k)}\,xp^2 + \underline{(-g/2)}\,x^3p^2$$

where the underlined coefficients make up the real elements of the L matrix for the truncation finally being considered. For the above, the first six basis vectors have been operated on by the Liouville operator. This produces the 6×6 truncation of the matrix representation of the operator $\mathcal{L}$, which we shall call L. This L matrix and its basis vectors are illustrated below:

	x	p	x^3	x^2p	xp^2	p^3
x	0	1/m	0	0	0	0
p	-k	0	-g/3!	0	0	0
x^3	0	0	0	3/m	0	0
x^2p	0	0	-k	0	2/m	0
xp^2	0	0	0	-2k	0	1/m
p^3	0	0	0	0	-3k	0

$= L$

where the term operated upon by $\mathcal{L}$ resides in the column to the left of the dotted vertical line at the left, and the basis for the N=6 truncation is exhibited in the top row above the dotted line, e.g.

$$\mathcal{L}\{x^2p\}_{N=5} = -k\,x^3 + (2/m)\,xp^2$$

which compares to

$$\mathcal{L}\{x^2p\} = -k\,x^3 + (2/m)\,xp^2 - (g/3!)\,x^5$$

for the full complement of terms produced when the $\mathcal{L}$ operator acts on the monomial x^2p. It is instructive to note that when $\mathcal{L}$ operates on the various vector monomials, the operator produces both old vector monomials and new vector monomials. The vector monomials of degree 3, for instance, are all given by individual terms of the binomial expression,

$$(x + p)^3 ,$$

whose four terms are independent of each other. Since $\mathcal{L}\{x^2p\}$ produces a term in x^5, it is anticipated that the next truncation should include the terms of

$$(x + p)^5 ,$$

which adds six more independent monomials of degree 5 to the set of basis vectors. This six member group of basis vectors brings the accumulated total of basis vectors to 12.

This process can be continued. The next complete truncation includes the terms of the binomial expansion

$$(x + p)^7 ,$$

which brings the accumulated total of basis vectors to 20.

The next complete truncation includes the terms of the binomial

$$(x + p)^9 ,$$

bringing the accumulated total of basis vectors to 30, the next includes the terms of

$$(x + p)^{11} ,$$

bringing the total to 42, and the next includes the terms of

$$(x + p)^{13} ,$$

bringing the accumulated total of basis vectors to 56. This process can obviously be continued indefinitely generating an infinite L matrix of coefficients. For the purpose of this investigation the independent basis was terminated at 56.

The structure of the 56×56 truncation of the L matrix is illustrated next. If the square matrices A, B, C, D, E, F, and K, each of even dimension and the rectangular matrices, G(n), which are a function of the constant g only and where n is odd, are introduced, the entire 56x56 truncated L matrix can be represented by the matrix blocked as follows

$$L = \left[\begin{array}{c|c|c|c|c|c|c} A & G(1) & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & B & G(3) & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & C & G(5) & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & D & G(7) & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & E & G(9) & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & F & G(11) \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & K \end{array}\right],$$

where 0 denotes the block zero matrix of appropriate dimensions and where the block matrices along the diagonal are given by

$$A = \begin{bmatrix} 0 & 1/m \\ -k & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 3/m & 0 & 0 \\ -k & 0 & 2/m & 0 \\ 0 & -2k & 0 & 1/m \\ 0 & 0 & -3k & 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 5/m & 0 & 0 & 0 & 0 \\ -k & 0 & 4/m & 0 & 0 & 0 \\ 0 & -2k & 0 & 3/m & 0 & 0 \\ 0 & 0 & -3k & 0 & 2/m & 0 \\ 0 & 0 & 0 & -4k & 0 & 1/m \\ 0 & 0 & 0 & 0 & -5k & 0 \end{bmatrix},$$

$$D = \begin{bmatrix} 0 & 7/m & 0 & 0 & 0 & 0 & 0 & 0 \\ -k & 0 & 6/m & 0 & 0 & 0 & 0 & 0 \\ 0 & -2k & 0 & 5/m & 0 & 0 & 0 & 0 \\ 0 & 0 & -3k & 0 & 4/m & 0 & 0 & 0 \\ 0 & 0 & 0 & -4k & 0 & 3/m & 0 & 0 \\ 0 & 0 & 0 & 0 & -5k & 0 & 2/m & 0 \\ 0 & 0 & 0 & 0 & 0 & -6k & 0 & 1/m \\ 0 & 0 & 0 & 0 & 0 & 0 & -7k & 0 \end{bmatrix},$$

$$E = \begin{bmatrix} 0 & 9/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k & 0 & 8/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2k & 0 & 7/m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3k & 0 & 6/m & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4k & 0 & 5/m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5k & 0 & 4/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6k & 0 & 3/m & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7k & 0 & 2/m & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -8k & 0 & 1/m \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9k & 0 \end{bmatrix},$$

$$F = \begin{bmatrix} 0 & 11/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k & 0 & 10/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2k & 0 & 9/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3k & 0 & 8/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4k & 0 & 7/m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5k & 0 & 6/m & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6k & 0 & 5/m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7k & 0 & 4/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -8k & 0 & 3/m & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9k & 0 & 2/m & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -10k & 0 & 1/m \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11k & 0 \end{bmatrix},$$

and

$$K = \begin{bmatrix} 0 & 13/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k & 0 & 12/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2k & 0 & 11/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3k & 0 & 10/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4k & 0 & 9/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5k & 0 & 8/m & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6k & 0 & 7/m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7k & 0 & 6/m & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -8k & 0 & 5/m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9k & 0 & 4/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -10k & 0 & 3/m & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11k & 0 & 2/m & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -12k & 0 & 1/m \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -13k & 0 \end{bmatrix},$$

and where the G(n) matrices are rectangular with n+1 rows and n+3 columns given by

$$G(1) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -g/3! & 0 & 0 & 0 \end{bmatrix}, \quad G(3) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -g/3! & 0 & 0 & 0 & 0 & 0 \\ 0 & -g/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & -g/2 & 0 & 0 & 0 \end{bmatrix}$$

$$G(5) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -g/3! & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -g/3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -g/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2g/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5g/6 & 0 & 0 & 0 \end{bmatrix},$$

$$G(7) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -g/3! & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -g/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -g/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2g/3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5g/6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7g/6 & 0 & 0 & 0 \end{bmatrix},$$

$$G(9) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -g/3! & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -g/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -g/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2g/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5g/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7g/6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4g/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3g/2 & 0 & 0 & 0 \end{bmatrix},$$

G(11) =

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-g/3! & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -g/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -g/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -2g/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -5g/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -7g/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -4g/3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3g/2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -5g/3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{11}{6}g & 0 & 0 & 0
\end{bmatrix},$$

If the 56×56 L matrix is partitioned in accordance with the foregoing block structure, where the blocks (partitions) are determined by each of the successive groups of terms obtainable from the j^{th} degree terms in x and p, then certain observations follow. First, if there are ρ partitions of the L matrix, the uppermost index of the l^{th} partition of the L matrix, denoted by kpartshn(l), is given by

$$\text{kpartshn}(l) = l\cdot(l+1) \quad \text{for } l = 1,\ldots,\rho$$

while the first index of the l^{th} partition of the L matrix is given by

$$\text{kpartshn}(l-1) + 1 \quad \text{for } l \geq 2 \ .$$

For the N = 56 truncation of L, $\rho = 7$.

The L matrix as defined above can be classified, for any truncation size, as an upper Hessenberg matrix of constant coefficients. An upper Hessenberg matrix, A, is a matrix for which

$$a_{i,j} = 0 \qquad \text{whenever } i > j + 1.$$

Most articles on Hessenberg matrices and the software routine HQR of the EISPACK guide (Smith et al 1976) assume that a Hessenberg matrix has been put into its upper Hessenberg form. The importance of the matrix being in upper Hessenberg form is that such a form facilitates the calculation of its eigenvalues. The routine HQR performs QR decomposition on an upper Hessenberg matrix, H, to produce the eigenvalues of H along the diagonal of the resultant decomposed matrix. The QR decomposition procedure is a numerical procedure upon the elements of a matrix that employs orthogonal iteration to produce a sequence of matrix iterates each of which can be factored into a product of a unitary matrix and an upper triangular matrix which, when multiplied in reverse order produces the subsequent matrix iterate. The matrix iterates converge to produce a triangular matrix which can be cleanly written as the sum of a diagonal matrix and a strictly upper triangular matrix (Schur decomposition). Eigenvalues then appear along the diagonal of the diagonal matrix as 1×1 or 2×2 submatrices depending upon whether the eigenvalues are

real or complex. This enables complete eigenvalue determination for the L matrix. The next task is the evaluation of the eigenvalue spectrum of any partitioned truncation of the L matrix. Let the eigenvalues of L be denoted, in general, by λ_i for i = 1,...N . Since L is a real general matrix, it is known that its eigenvalues occur in complex conjugate pairs. This is because the characteristic polynomial has real coefficients, is of even degree, and contains only even powers of λ. When the constant k is positive, the eigenvalues occur as complex conjugate pairs with zero real part. When the constant k is negative, the eigenvalues occur in real pairs with members having opposite sign of each other. For what follows, it is assumed that the constant k is positive. Then if the eigenvalue spectrum of L is denoted by $\sigma(\lambda)$, the structure of $\sigma(\lambda)$ for the N=56 truncation of the L matrix is given in Table 3.1 below. Appendix A provides the FORTRAN source used in obtaining the eigenvalues of the 56×56 truncation of the L matrix. Numerical values of the necessary machine epsilons required to form the floating point arithmetic system used by routines of the Eispack guide were obtained by exercising the subroutine found in Appendix B of Cody and Waite (Cody and Waite, 1980; 258-264). Table 3.1 is arranged in accordance with the partition structure of L, which was discussed earlier. In

Table 3.1 the value of the imaginary part of the smallest magnitude eigenvalue is denoted by β.

The detailed structure of $\sigma(\lambda)$ for the L matrix is apparent. The multiplicities of the eigenvalue pairs are as follows:

Eigenvalue Pair	$\pm i\beta$	$\pm 3i\beta$	$\pm 5i\beta$	$\pm 7i\beta$	$\pm 9i\beta$	$\pm 11i\beta$	$\pm 13i\beta$
multiplicities	7	6	5	4	3	2	1

Extension to the infinite matrix for L is obvious.

The value of β depends upon the values of m, k, and g. Changing the sign of the constant k appears only to affect the conversion of the eigenvalues from complex pairs to real oppositely signed pairs.

We have seen that the eigenvalue spectrum plays an important role in the approximation procedure, e.g. in the Putzer method the eigenvalue spectrum determines the form of the functions $r_K(t)$. The functions $r_K(t)$ have been calculated analytically for the eigenvalue spectrum belonging to L for the 2×2 and the 6×6 truncations for the Hamiltonian under study. The expressions for these functions are listed below for these truncations. For the 2×2 truncation the $r_K(t)$ functions are

$$r_1(t) = e^{i\omega t},$$

$$r_2(t) = \frac{1}{\omega} \sin(\omega t),$$

Table 3.1

Eigenvalue spectral structure for perturbed SHO

$m = 2.0 \quad k = 1.0 \quad g = 1.0$

$\beta = \sqrt{(k/m)}$

partition number	N	eigenvalue	
		real part	imag part
1	2	0	$\pm 1\beta$
2	6	0	$\pm 3\beta$ $\pm\beta$
3	12	0	$\pm 5\beta$ $\pm 3\beta$ $\pm\beta$
4	20	0	$\pm 7\beta$ $\pm 5\beta$ $\pm 3\beta$ $\pm\beta$
5	30	0	$\pm 9\beta$ $\pm 7\beta$ $\pm 5\beta$ $\pm 3\beta$ $\pm\beta$
6	42	0	$\pm 11\beta$ $\pm 9\beta$ $\pm 7\beta$ $\pm 5\beta$ $\pm 3\beta$ $\pm\beta$
7	56	0	$\pm 13\beta$ $\pm 11\beta$ $\pm 9\beta$ $\pm 7\beta$ $\pm 5\beta$ $\pm 3\beta$ $\pm\beta$

where $\omega = \beta = \sqrt{(k/m)}$. The eigenvalue pairs for the 6×6 truncation, the first case in which repeated eigenvalues appear, are $\pm i\omega$, $\pm i\mu$, and $\pm i\omega$ ($\omega = \beta$, $\mu = 3\beta$), and the $r_K(t)$ functions are the two functions above plus

$$r_3(t) = -\frac{1}{(\omega^2 - \mu^2)}\cos(\omega t) - \frac{i\,\mu}{\omega(\omega^2 - \mu^2)}\sin(\omega t) + \frac{1}{\omega^2 - \mu^2}\left[\cos(\mu t) + i\sin(\mu t)\right],$$

$$r_4(t) = \frac{-1}{\omega(\omega^2 - \mu^2)}\sin(\omega t) + \frac{1}{\mu(\omega^2 - \mu^2)}\sin(\mu t)$$

$$r_5(t) = \frac{i}{2\omega(\omega^2 - \mu^2)}\,t\,e^{i\omega t} - \frac{i}{2\omega^2(\omega^2 - \mu^2)}\sin(\omega t) + \frac{1}{(\omega^2 - \mu^2)^2}\cos(\mu t) + \frac{i\,\omega}{\mu(\omega^2 - \mu^2)^2}\sin(\mu t) + \frac{1}{(\omega^2 - \mu^2)^2}\,e^{i\omega t},$$

$$r_6(t) = \frac{1}{2\omega^2(\omega^2 - \mu^2)}\,t\cos(\omega t) - \frac{1}{2\omega^3(\omega^2 - \mu^2)}\sin(\omega t) - \frac{1}{\omega(\omega^2 - \mu^2)^2}\sin(\omega t) + \frac{1}{\mu(\omega^2 - \mu^2)^2}\sin(\mu t).$$

In order to obtain numerical results of those r(t) functions whose index is higher than 6, a numerical

integrator was written to calculate the value of the k^{th} r function at a given time τ. The numerical integrator employed is Gill's implementation of the Runge-Kutta procedure. See Appendix C of Viscous Fluid Flow (White 1974) for a discussion of this integrator. This integrator is a fourth order Runge-Kutta procedure. A FORTRAN source listing to implement this integrator to obtain numerically an exact solution to the perturbed simple harmonic oscillator is provided in Appendix C.

A discussion of the P matrices is given next. The P matrices depend on the eigenvalues. They are calculated by using a recursion relationship in which only the value of λ_j is needed to calculate P_j if the previous P_k matrices are known ($0 \leq k \leq j-1$). To save on computer storage space only the first two rows of each P matrix is stored, since the approximation is being used to calculate the first two basis vectors x and p. Appendix B contains a listing of the FORTRAN source for the 20×20 truncation of the L matrix.

For the 6×6 truncation of the L matrix the Putzer algorithm has been used to determine analytically the expressions for the approximated x(t) and p(t). Attention is now turned to the analytical expression for e^{Lt} constructed in accordance with the Putzer algorithm for the 6×6 L matrix. The L matrix for this truncation is

$$L = \begin{bmatrix} 0 & 1/m & 0 & 0 & 0 & 0 \\ -k & 0 & -g/3! & 0 & 0 & 0 \\ 0 & 0 & 0 & 3/m & 0 & 0 \\ 0 & 0 & -k & 0 & 2/m & 0 \\ 0 & 0 & 0 & -2k & 0 & 1/m \\ 0 & 0 & 0 & 0 & -3k & 0 \end{bmatrix},$$

The eigenvalues are $\pm i\omega$ with multiplicity 2 and, $\pm i3\omega$ with multiplicity 1, where $\omega = \sqrt{k/m}$. That there is no g dependence in the eigenvalues for this truncation can be verified by simple calculation. The $r_K(t)$ functions are

$$r_1(t) = \cos(\omega t) + i\sin(\omega t),$$

$$r_2(t) = \frac{1}{\omega}\sin(\omega t),$$

$$r_3(t) = \frac{1}{8\omega^2}\left\{\cos(\omega t) - \cos(3\omega t)\right\} + i\,\frac{1}{8\omega^2}\left\{3\sin(\omega t) - \sin(3\omega t)\right\},$$

$$r_4(t) = \frac{1}{8\omega^3}\left\{\sin(\omega t) - \frac{1}{3}\sin(3\omega t)\right\},$$

$$r_5(t) = i\,\frac{1}{64\omega^4}\left\{3\sin(\omega t) + \frac{1}{3}\sin(3\omega t)\right\} - \frac{1}{64\omega^4}\left\{\cos(\omega t) - \cos(3\omega t)\right\} - \frac{i}{16\omega^3}t\cos(\omega t) + \frac{1}{16\omega^3}t\sin(\omega t),$$

and

$$r_6(t) = \frac{1}{64\omega^5}\left[3\sin(\omega t) + \frac{1}{3}\sin(3\omega t)\right] - \frac{1}{16\omega^4}\, t\cos(\omega t).$$

The six P matrices are $P_0 = \mathbf{I}$, the 6×6 identity matrix, and

$$P_1 = \begin{bmatrix} -i\omega & 1/m & 0 & 0 & 0 & 0 \\ -k & -i\omega & -g/3! & 0 & 0 & 0 \\ 0 & 0 & -i\omega & 3/m & 0 & 0 \\ 0 & 0 & -k & -i\omega & 2/m & 0 \\ 0 & 0 & 0 & -2k & -i\omega & 1/m \\ 0 & 0 & 0 & 0 & -3k & -i\omega \end{bmatrix},$$

$$P_2 = \begin{bmatrix} 0 & 0 & -g/(m3!) & 0 & 0 & 0 \\ 0 & 0 & 0 & -g/2m & 0 & 0 \\ 0 & 0 & -2\omega^2 & 0 & 6/m^2 & 0 \\ 0 & 0 & 0 & -6\omega^2 & 0 & 2/m^2 \\ 0 & 0 & 2k^2 & 0 & -6\omega^2 & 0 \\ 0 & 0 & 0 & 6k^2 & 0 & -2\omega^2 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 0 & 0 & ig\omega/2m & -g/2m^2 & 0 & 0 \\ 0 & 0 & g\omega^2/2 & i3g\omega/2m & -g/m^2 & 0 \\ 0 & 0 & i6\omega^3 & -18\omega^2/m & -i18\omega/m^2 & 6/m^3 \\ 0 & 0 & 6k\omega^2 & i18\omega^3 & -18\omega^2/m & -i6\omega/m^2 \\ 0 & 0 & -i6k^2\omega & 18k\omega^2 & i18\omega^3 & -6\omega^2/m \\ 0 & 0 & -6k^3 & -i18k^2\omega & 18k\omega^2 & i6\omega^3 \end{bmatrix},$$

$$P_4 = \begin{bmatrix} 0 & 0 & -g\omega^2/m & 0 & -g/m^3 & 0 \\ 0 & 0 & 0 & -g\omega^2/m & 0 & -g/m^3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and

$$P_5 = \begin{bmatrix} 0 & 0 & ig\omega^3/m & -g\omega^2/m^2 & ig\omega/m^3 & -g/m^4 \\ 0 & 0 & g\omega^4 & ig\omega^3/m & g\omega^2/m^2 & ig\omega/m^3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Performing the Putzer sum as indicated in equation 2.1 where n = 6, e^{Lt} can be written as

$$e^{Lt} = \begin{bmatrix} \cos(\omega t) & \frac{1}{m\omega}\sin(\omega t) & e_{13} & e_{14} & e_{15} & e_{16} \\ -m\omega\sin(\omega t) & \cos(\omega t) & e_{23} & e_{24} & e_{25} & e_{26} \\ 0 & 0 & e_{33} & e_{34} & e_{35} & e_{36} \\ 0 & 0 & e_{43} & e_{44} & e_{45} & e_{46} \\ 0 & 0 & e_{53} & e_{54} & e_{55} & e_{56} \\ 0 & 0 & e_{63} & e_{64} & e_{65} & e_{66} \end{bmatrix},$$

where

$$e_{13} = -\frac{1}{192}\frac{g}{m\omega^2}\left\{\cos(\omega t) - \cos(3\omega t)\right\} - \frac{1}{16}\frac{g}{m\omega^2}\omega t\sin(\omega t),$$

$$e_{14} = -\frac{1}{64}\frac{g}{m^2\omega^3}\left\{7\sin(\omega t) - \sin(3\omega t)\right\} + \frac{1}{16}\frac{g}{m^2\omega^3}\omega t\cos(\omega t) \quad ,$$

$$e_{15} = \frac{1}{64}\frac{g}{m^3\omega^4}\left\{\cos(\omega t) - \cos(3\omega t)\right\} - \frac{1}{16}\frac{g}{m^3\omega^4}\omega t\sin(\omega t) \quad ,$$

$$e_{16} = -\frac{1}{64}\frac{g}{m^4\omega^5}\left\{3\sin(\omega t) + \frac{1}{3}\sin(3\omega t)\right\} + \frac{1}{16}\frac{g}{m^4\omega^5}\omega t\cos(\omega t) \quad ,$$

$$e_{23} = -\frac{1}{192}\frac{g}{\omega}\left\{11\sin(\omega t) + 3\sin(3\omega t)\right\} - \frac{1}{16}\frac{g}{\omega}\omega t\cos(\omega t) \quad ,$$

$$e_{24} = -\frac{3}{64}\frac{g}{m\omega^2}\left\{\cos(\omega t) - \cos(3\omega t)\right\} - \frac{1}{16}\frac{g}{m\omega^2}\omega t\sin(\omega t) \quad ,$$

$$e_{25} = -\frac{1}{64}\frac{g}{m^2\omega^3}\left\{5\sin(\omega t) - 3\sin(3\omega t)\right\} - \frac{1}{16}\frac{g}{m^2\omega^3}\omega t\cos(\omega t) \quad ,$$

$$e_{26} = \frac{1}{64}\frac{g}{m^3\omega^4}\left\{\cos(\omega t) - \cos(3\omega t)\right\} - \frac{1}{16}\frac{g}{m^3\omega^4}\,\omega t \sin(\omega t) \quad ,$$

$$e_{33} = e_{66} = \frac{1}{4}\left\{3\cos(\omega t) + \cos(3\omega t)\right\} \quad ,$$

$$e_{44} = e_{55} = \frac{1}{4}\left\{\cos(\omega t) + 3\cos(3\omega t)\right\} \quad ,$$

$$e_{43} = -\frac{m\omega}{4}\left\{\sin(\omega t) + \sin(3\omega t)\right\} \quad ,$$

$$e_{34} = \frac{3}{4m\omega}\left\{\sin(\omega t) + \sin(3\omega t)\right\} \quad ,$$

$$e_{54} = \frac{m\omega}{4}\left\{\sin(\omega t) - 3\sin(3\omega t)\right\} \quad ,$$

$$e_{45} = -\frac{1}{4m\omega}\left\{\sin(\omega t) - 3\sin(3\omega t)\right\} \quad ,$$

$$e_{65} = -\frac{3m\omega}{4}\left\{\sin(\omega t) + \sin(3\omega t)\right\} \quad ,$$

$$e_{56} = \frac{1}{4m\omega}\left\{\sin(\omega t) + \sin(3\omega t)\right\} \quad ,$$

$$e_{53} = \frac{(m\omega)^2}{4}\left\{\cos(\omega t) - \cos(3\omega t)\right\} \quad ,$$

$$e_{35} = \frac{3}{4(m\omega)^2}\left\{\cos(\omega t) - \cos(3\omega t)\right\},$$

$$e_{54} = \frac{3(m\omega)^2}{4}\left\{\cos(\omega t) - \cos(3\omega t)\right\},$$

$$e_{46} = \frac{1}{4(m\omega)^2}\left\{\cos(\omega t) - \cos(3\omega t)\right\},$$

$$e_{53} = -\frac{(m\omega)^3}{4}\left\{3\sin(\omega t) - \sin(3\omega t)\right\},$$

$$e_{36} = \frac{1}{4(m\omega)^3}\left\{3\sin(\omega t) - \sin(3\omega t)\right\}.$$

Thus, when the calculation

$$\vec{U}(t) = e^{Lt}\,\vec{U}(t=0),$$

is performed, the results for the first two elements of $\vec{U}(t)$ yield

$$x(t) = x_o\cos(\omega t) + p_o\frac{1}{m\omega}\sin(\omega t)$$
$$+ x_o^3\left\{-\frac{1}{192}\frac{g}{m\omega^2}\left[\cos(\omega t) - \cos(3\omega t)\right] - \frac{1}{16}\frac{g}{m\omega^2}\omega t\sin(\omega t)\right\}$$
$$+ x_o^2 p_o\left\{-\frac{1}{64}\frac{g}{m^2\omega^3}\left[7\sin(\omega t) - \sin(3\omega t)\right] + \frac{1}{16}\frac{g}{m^2\omega^3}\omega t\cos(\omega t)\right\}$$

$$+ x_o p_o^2 \left\{ \frac{1}{64} \frac{g}{m^3\omega^4} \left[\cos(\omega t) - \cos(3\omega t) \right] - \frac{1}{16} \frac{g}{m^3\omega^4} \omega t \sin(\omega t) \right\}$$

$$+ p_o^3 \left\{ - \frac{1}{64} \frac{g}{m^4\omega^5} \left[3 \sin(\omega t) + \frac{1}{3} \sin(3\omega t) \right] + \frac{1}{16} \frac{g}{m^4\omega^5} \omega t \cos(\omega t) \right\} ,$$

and

$$p(t) = -x_o m\omega \sin(\omega t) + p_o \cos(\omega t)$$

$$+ x_o^3 \left\{ - \frac{1}{192} \frac{g}{\omega} \left[11 \sin(\omega t) + 3 \sin(3\omega t) \right] - \frac{1}{16} \frac{g}{\omega} \omega t \cos(\omega t) \right\}$$

$$+ x_o^2 p_o \left\{ - \frac{3}{64} \frac{g}{m\omega^2} \left[\cos(\omega t) - \cos(3\omega t) \right] - \frac{1}{16} \frac{g}{m\omega^2} \omega t \sin(\omega t) \right\}$$

$$+ x_o p_o^2 \left\{ - \frac{1}{64} \frac{g}{m^2\omega^3} \left[5 \sin(\omega t) - 3 \sin(3\omega t) \right] - \frac{1}{16} \frac{g}{m^2\omega^3} \omega t \cos(\omega t) \right\}$$

$$+ p_o^3 \left\{ \frac{1}{64} \frac{g}{m^3\omega^4} \left[\cos(\omega t) - \cos(3\omega t) \right] - \frac{1}{16} \frac{g}{m^3\omega^4} \omega t \sin(\omega t) \right\} .$$

For the 6×6 truncation, at least, the first terms that are part of an expansion in time are now apparent. For x(t) the time dependence of these non-oscillatory terms goes as

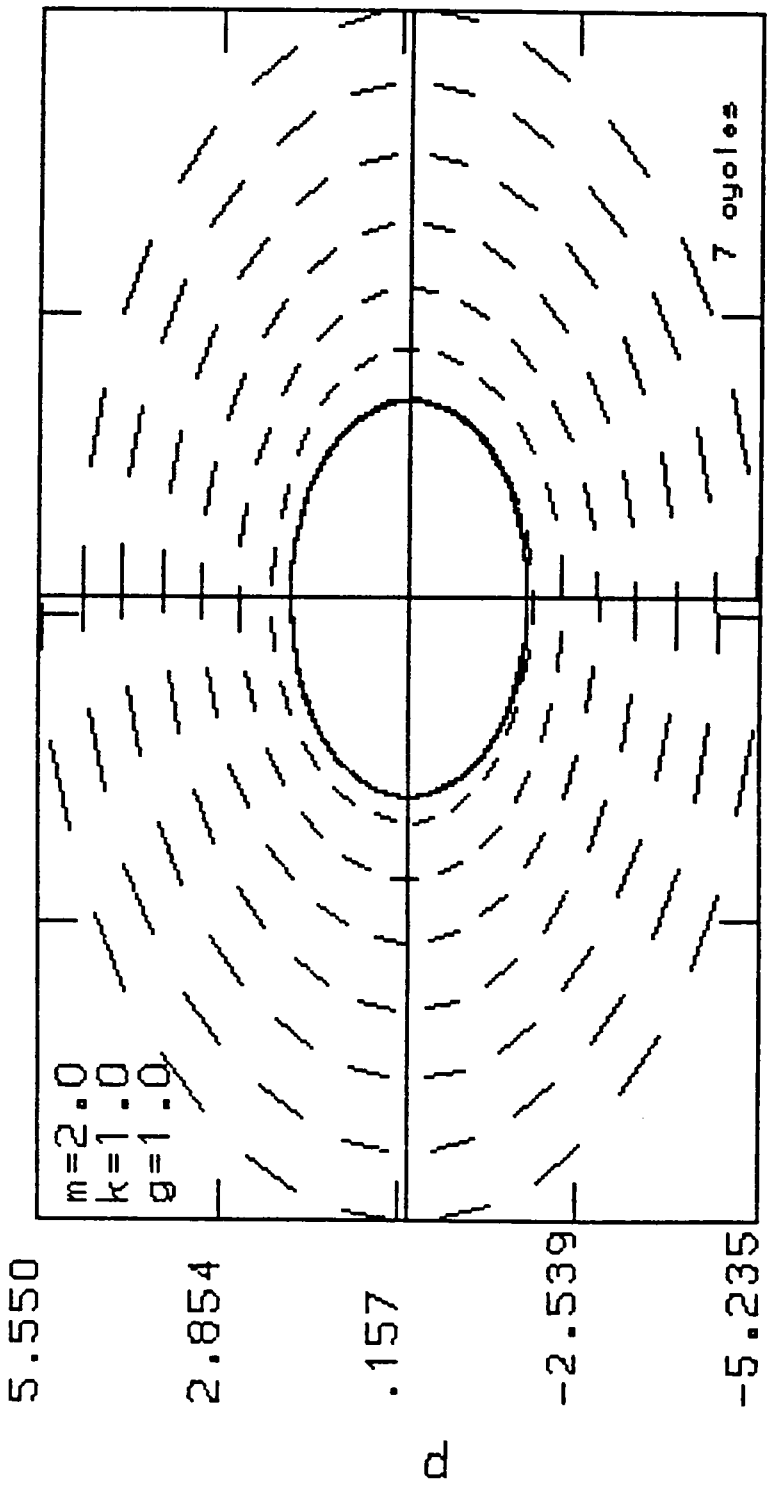
$$x_{non\text{-}osc}(t) \sim \left[-\frac{g}{16}\frac{\sin(\omega t)}{m\omega}\left\{x_o^3 + \frac{1}{(m\omega)^2}x_o p_o^2\right\} + \frac{g}{16}\frac{\cos(\omega t)}{(m\omega)^2}\left\{x_o^2 p_o + \frac{1}{(m\omega)^2}p_o^3\right\}\right] t \ ,$$

while for p(t)

$$p_{non\text{-}osc}(t) \sim \left[-\frac{g}{16}\cos(\omega t)\left\{x_o^3 + \frac{1}{(m\omega)^2}x_o p_o^2\right\} - \frac{g}{16}\frac{\sin(\omega t)}{m\omega}\left\{x_o^2 p_o + \frac{1}{(m\omega)^2}p_o^3\right\}\right] t \ .$$

Figure 3.1 shows, for 7 cycles, a phase space plot which compares x(t) and p(t), calculated analytically using e^{Lt} in the 6×6 truncation of L, with the exact x(t) and p(t) as obtained from the numerical Runge-Kutta-Gill (RKG) numerical integration of the differential equations produced from Hamilton's equations. Table 3.2 provides a summary of the standard deviations for the differences in the position as well as the momentum on a per cycle basis.

The statistics contained in this dissertation are calculated throughout with the following formulation.



PHASE PLOT exact & anal exp(Lt) 6x6

Figure 3.1

Table 3.2

Statistics for Δx and Δp for analytically calculated e^{-Lt} for 6×6 truncation and exact solution

$m = 2.0, \quad k = 1.0, \quad g = 1.0$

number of cycles	Number of data points	1 σ values	
		position	momentum
1	44	.130128	.261736
2	87	.368784	.619035
3	130	.655199	1.047480
4	174	.981892	1.542162
5	218	1.329919	2.054819
6	262	1.679093	2.556741
7	300	1.994893	2.920290

Differences in both position and momentum are calculated for each data point using

$$(\Delta x)_i = x_i - x'_i$$

$$(\Delta p)_i = p_i - p'_i ,$$

where the primed variable refers to the value from the approximation scheme. The standard deviation for Δx is then calculated using the usual formulation

$$\sigma_{\Delta x} = \left\{ \frac{\sum_{i=1}^{n}((\Delta x)_i^2) - \frac{\left[\sum_{i=1}^{n}(\Delta x)_i\right]^2}{n}}{n-1} \right\}^{1/2} ,$$

with an analogous expression for Δp.

Figure 3.2 is a phase space plot, again for the 7 cycle case, of $\Delta x(t)$ from the two positional calculations versus $\Delta p(t)$ from the two momentum calculations as depicted in Figure 3.1 . A check of e^{Lt} as analytically calculated with the Putzer algorithm for the 6×6 truncation of L is provided by comparing the phase space plot of x(t) and p(t) as derived from it, with the phase space plot of x(t) and p(t) obtained from the numerically integrated Putzer method using the 6×6 truncation of the L matrix. These phase space plots are presented in Figure 3.3 , where the solid line is a plot of the analytical results while

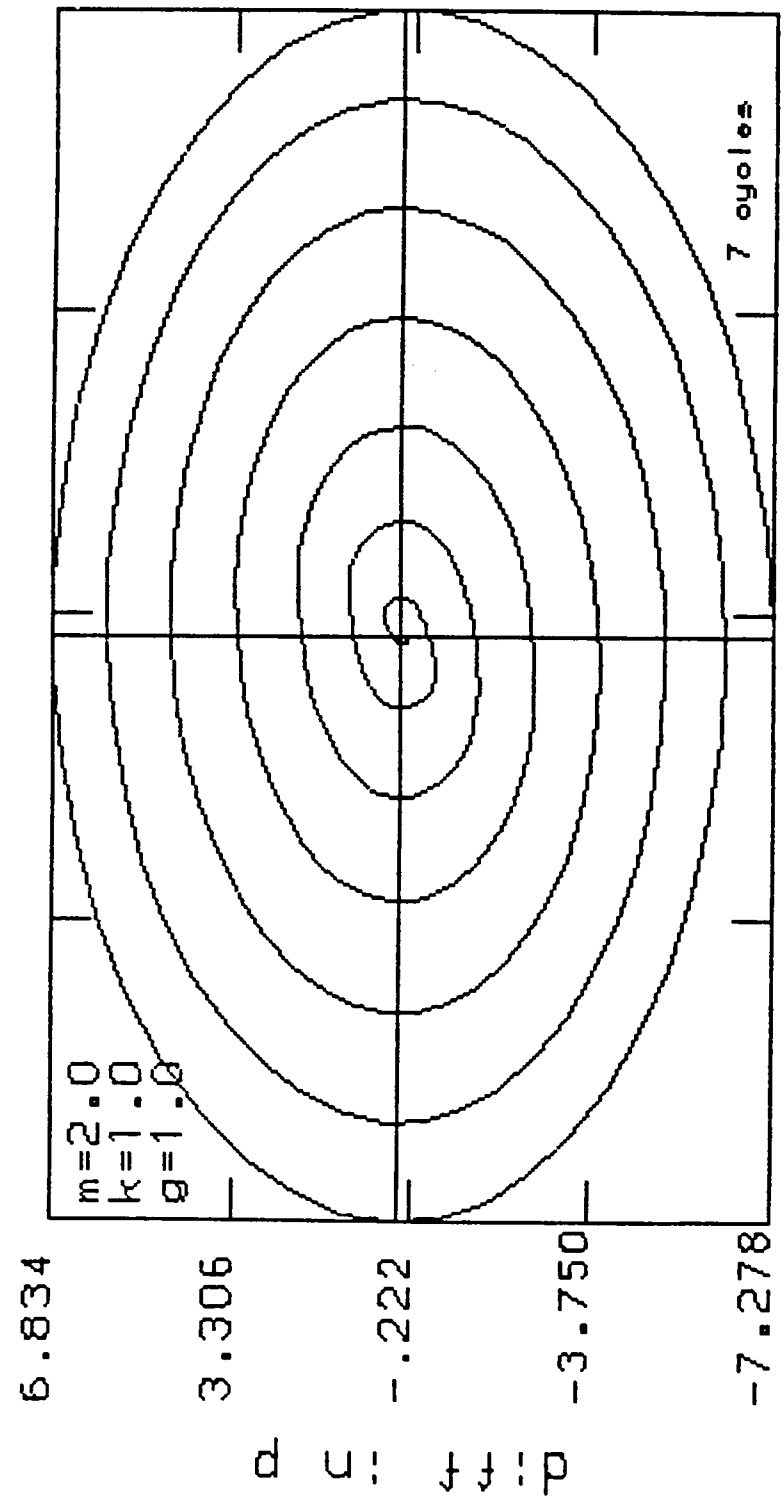
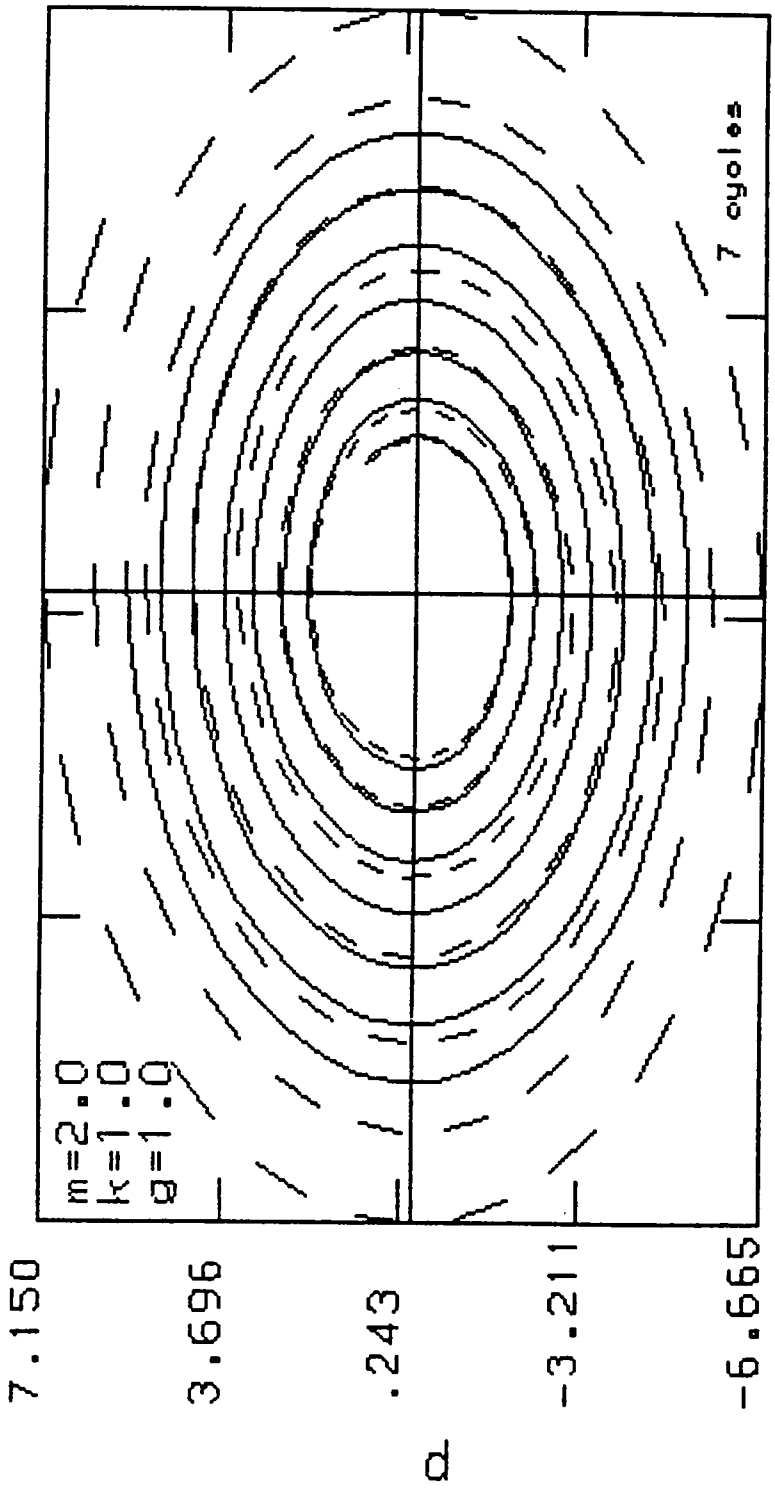

6.834
3.306
-.222
-3.750
-7.278
diff in p
m=2.0
k=1.0
g=1.0
7 cycles
-4.527
-2.183
.161
2.506
4.850
diff in x
PHASE DIFF exact & anal exp(Lt) 6x6

Figure 3.2

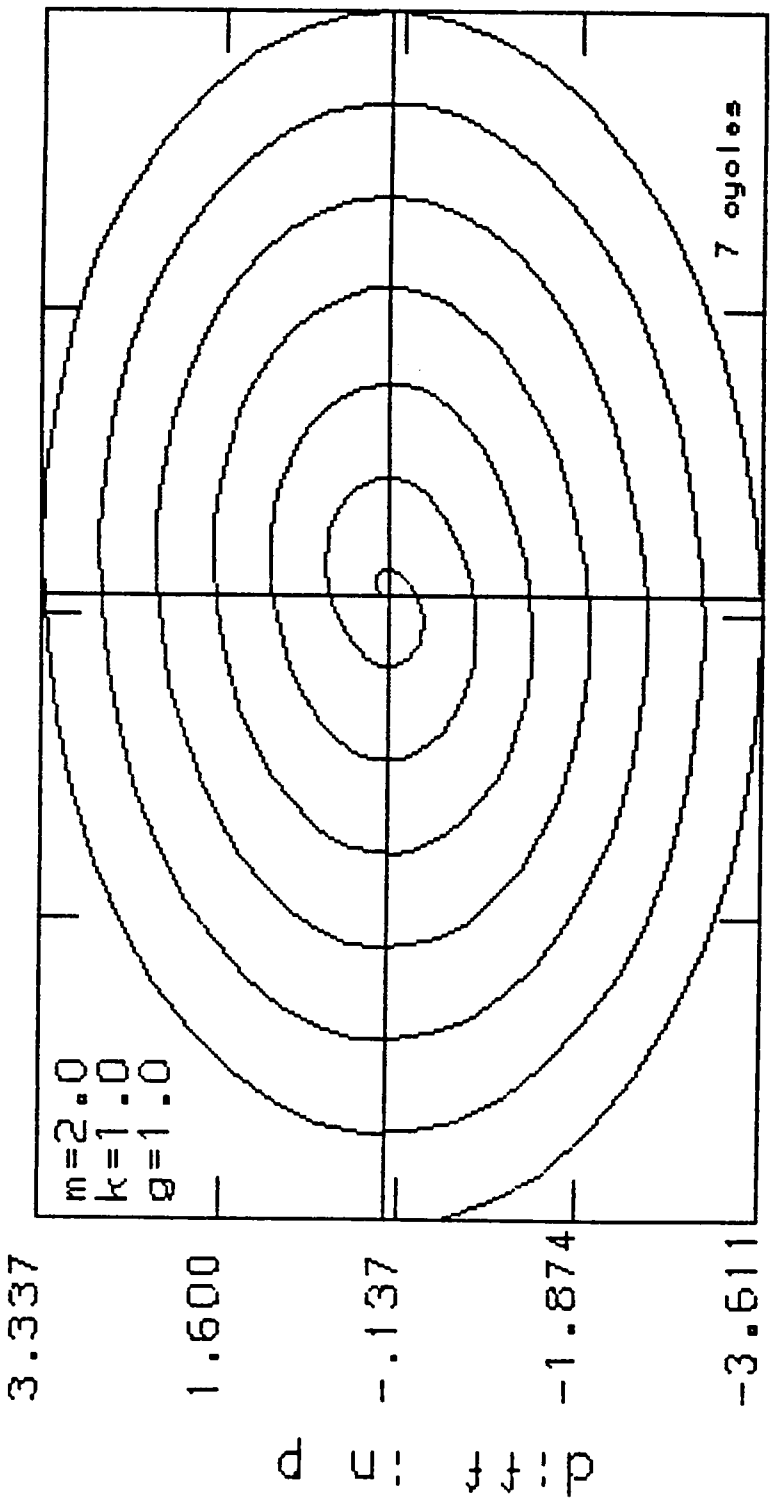


PHASE PLOT anal exp(Lt) & 6x6 approx

Figure 3.3

the dashed line is the plot of the numerically integrated approximation using the Putzer algorithm. Figure 3.4 exhibits the differences in phase space for the two techniques calculating x(t) and p(t) of Figure 3.3 . Obviously there is some error present in either the analytically calculated results or the numerical technique. It is believed that the error is in the numerical results. The analytical results have been checked by two different approaches on two separate occasions with the same results obtained in both instances. The magnitude of x(t) and p(t) of the analytical results are less than the results of the numerical results for more remote cycles. Table 3.3 provides a summary of the statistics associated with the various cycles contained in Figure 3.3.

Figure 3.5 presents the phase space plot of the exact RKG solution (solid line) of the motion and the plot (dashed line) of the approximation using the Putzer algorithm with the 6×6 truncation of the L matrix and RKG integration of the $r_K(t)$ functions. Seven cycles of data are presented. Figure 3.6 is a plot in phase space of the differences in x and p between the exact solution and the approximation using the Putzer algorithm for the 6×6 truncation of L over the same 7 cycles. Table 3.4 lists the standard deviations for both the position and the momentum on a cycle by cycle basis for the data presented



PHASE DIFF anal exp(Lt) & 6x6 approx

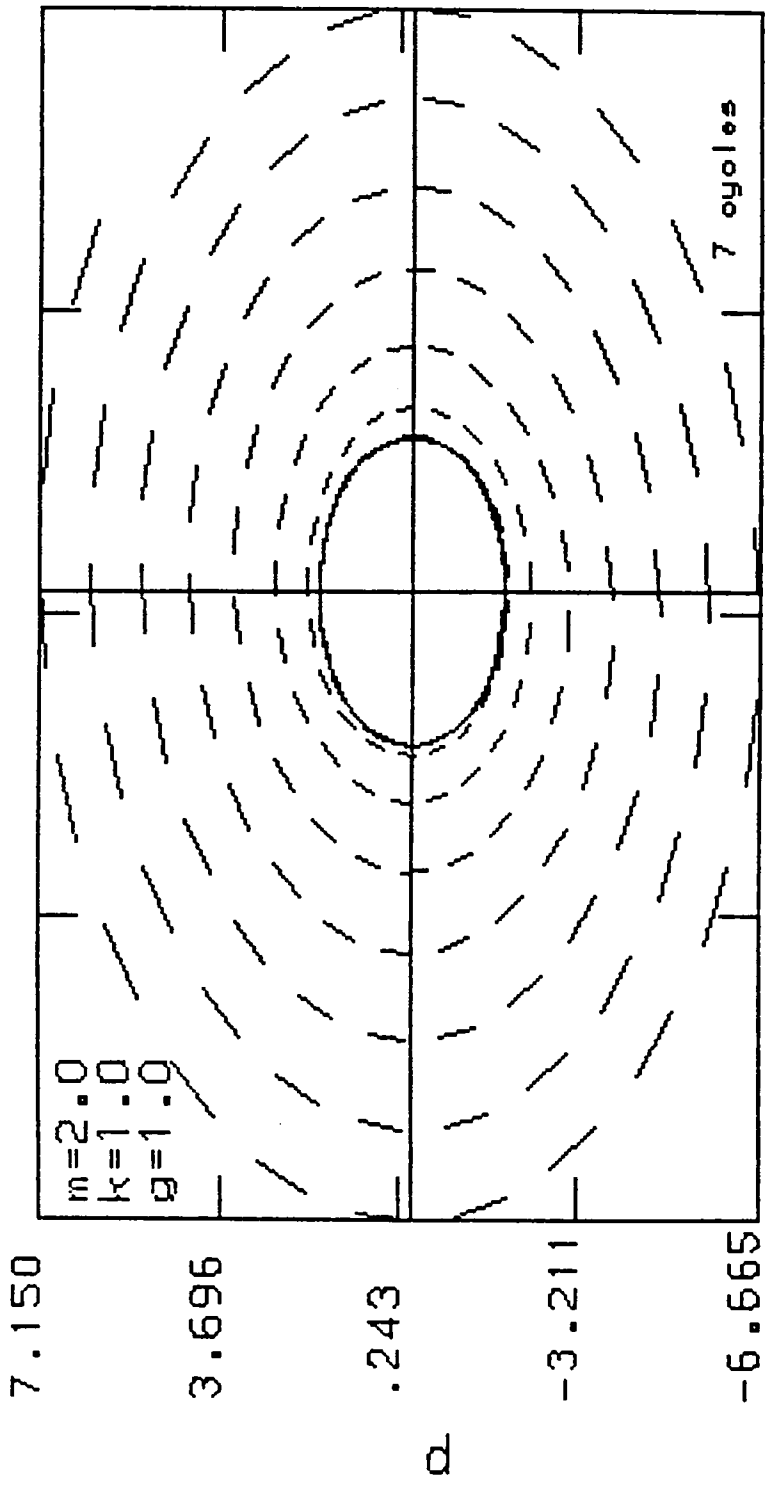
Figure 3.4

Table 3.3

Statistics for Δx and Δp for analytic e^{Lt} and numerical e^{Lt} for 6×6 L truncation

$m = 2.0, \quad k = 1.0, \quad g = 1.0$

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.149060	.193295
2	84	.316591	.399348
3	127	.473586	.620006
4	171	.632021	.845761
5	215	.789840	1.071779
6	259	.947231	1.298060
7	300	1.078966	1.530432



PHASE PLOT exact & 6x6 approx PUTZER

Figure 3.5

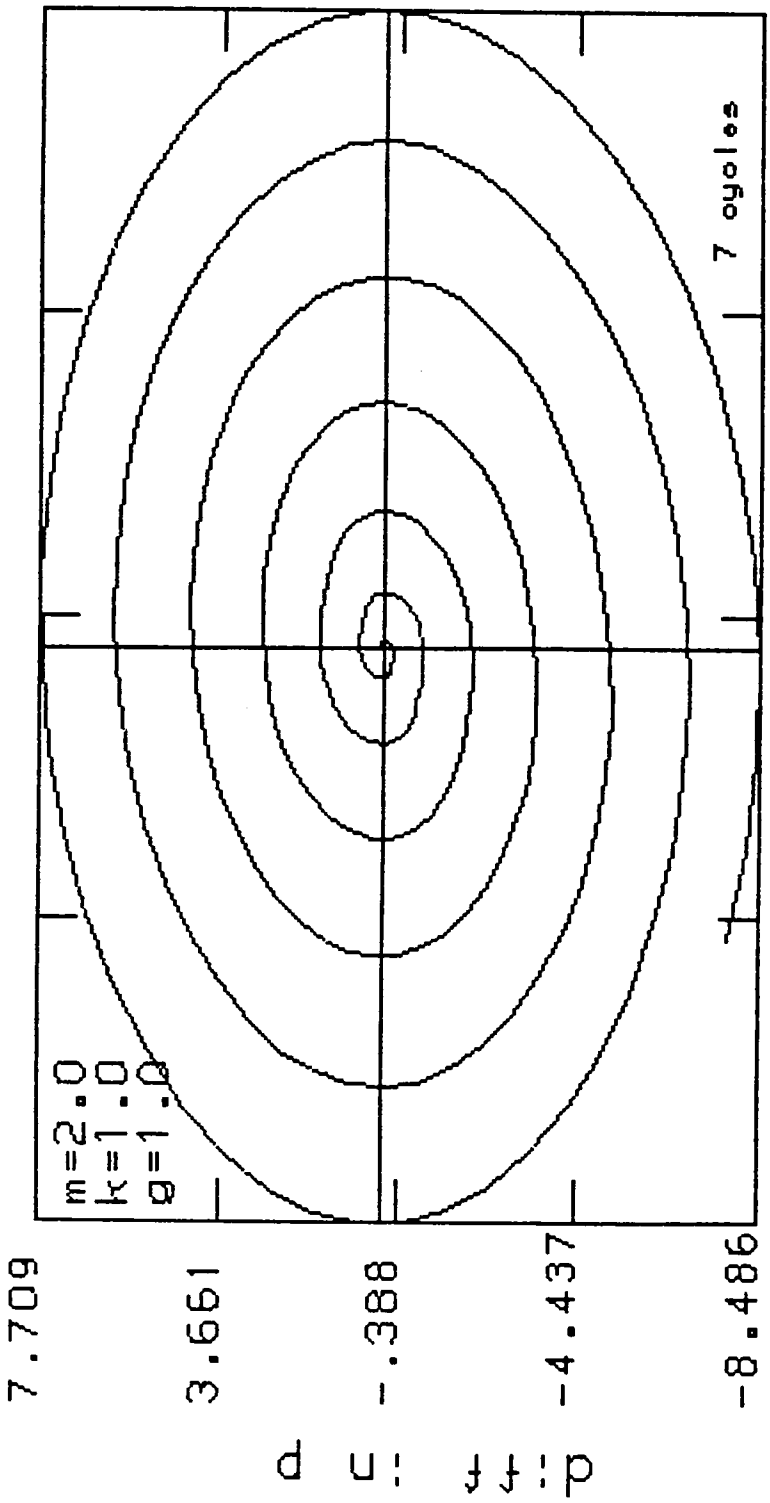

PHASE DIFF exact & 6x6 approx PUTZER
m=2.0
k=1.0
g=1.0
7 cycles
diff in p
7.709
3.661
-.388
-4.437
-8.486
-4.982
-2.348
.285
2.919
5.552
diff in x

Figure 3.6

Table 3.4

Statistics for Δx and Δp for exact solution and 6x6 approximation using Putzer algorithm with RKG integration

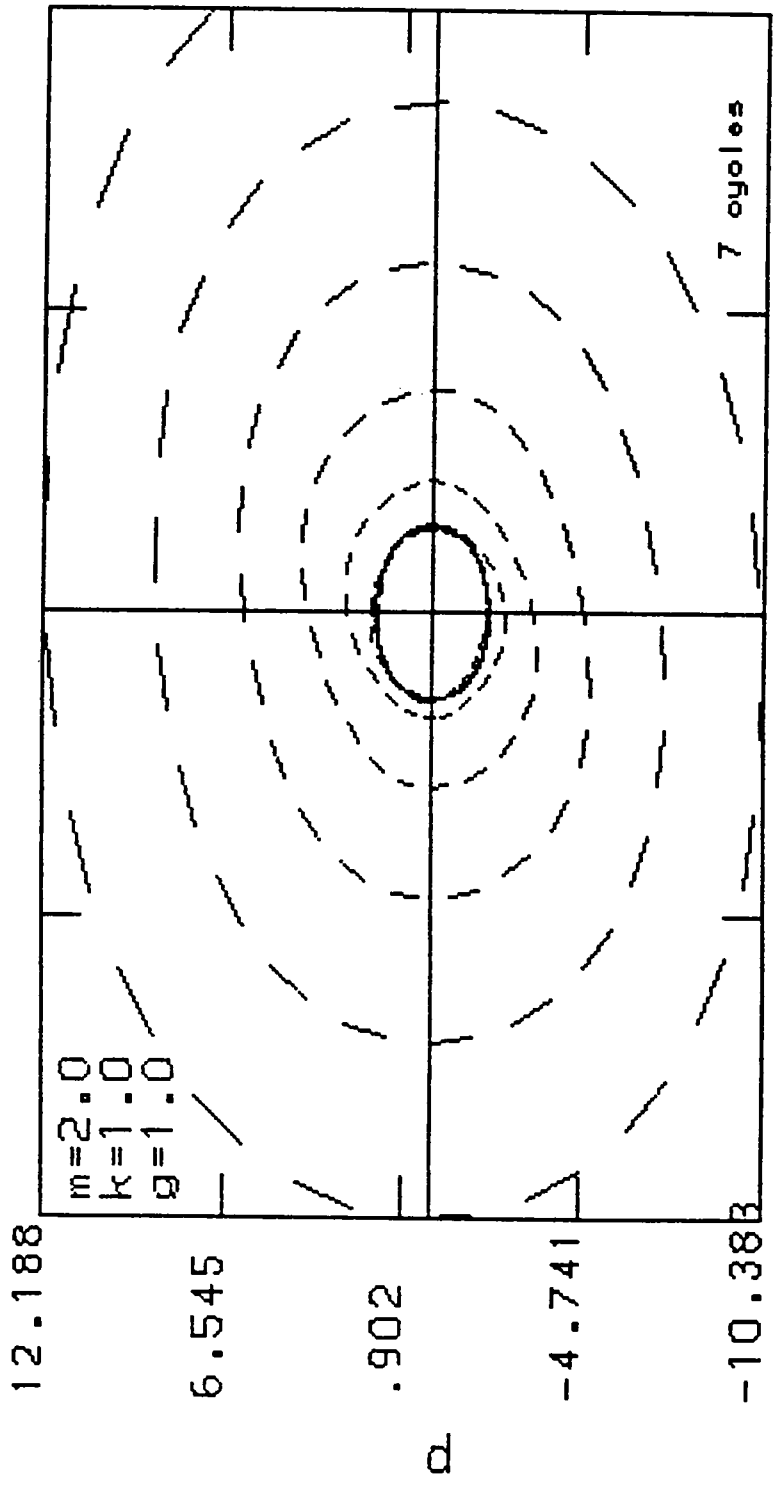
$m = 2.0, \quad k = 1.0, \quad g = 1.0$

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.070960	.107610
2	84	.212665	.365961
3	127	.462109	.755445
4	171	.801873	1.271066
5	215	1.205238	1.870774
6	259	1.647972	2.516208
7	300	2.074728	3.121994

in Figure 3.5 .

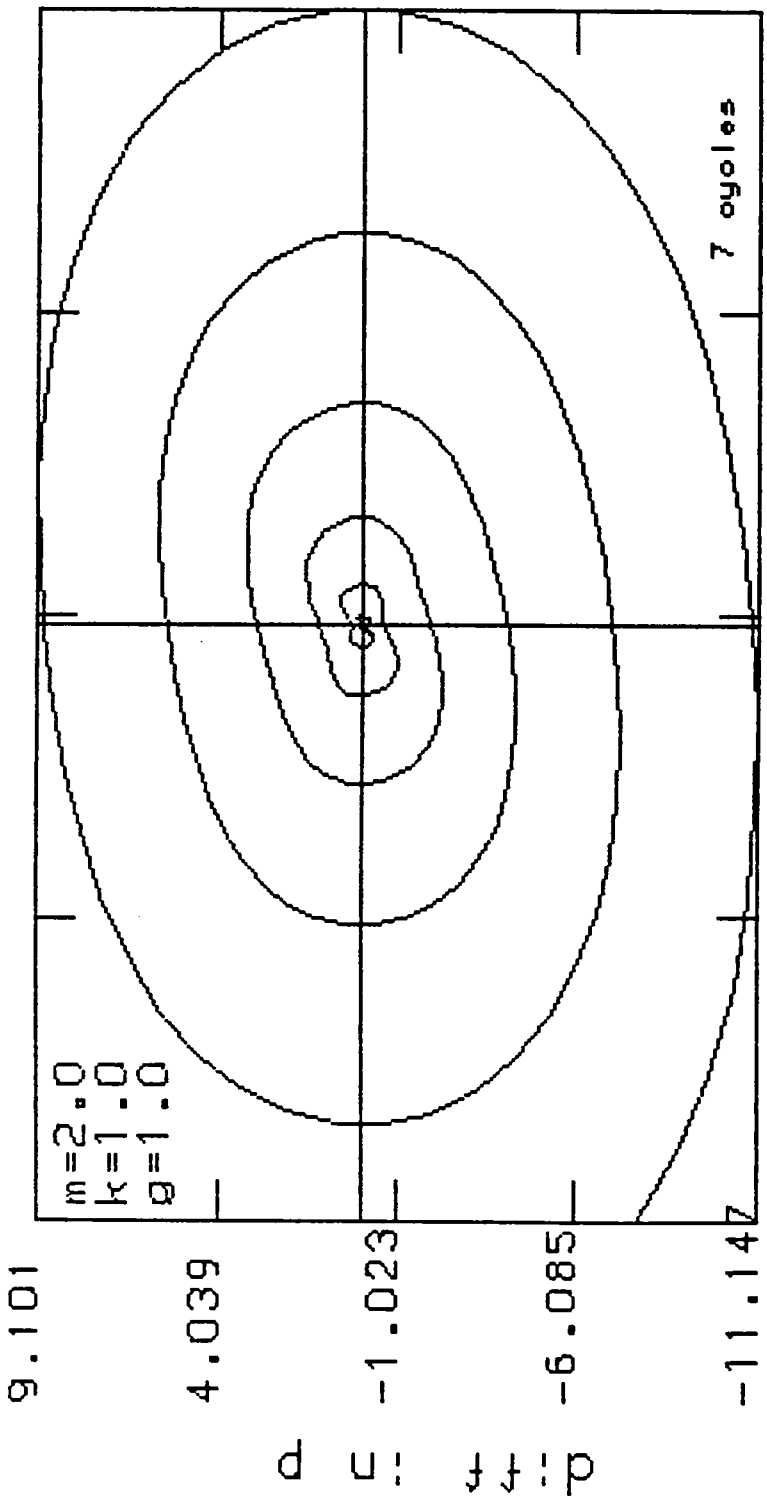
Figure 3.7 exhibits, for 7 cycles of data, the phase space plots of the exact solution and the approximation using the Putzer algorithm for the 12×12 truncation of the L matrix. The phase space differences in x and p for this case are plotted in Figure 3.8 . The associated statistics for Figure 3.8 are presented in Table 3.5 on a cycle by cycle basis. The plots for two cycles of data for the 12×12 case, presented in Figure 3.9 for the phase plot of the positions and momenta, and in Figure 3.10 for the differences of the positions and the momenta in phase space, show a reasonably good agreement between the approximation and the exact solution.

The data produced by the 20×20 truncation of the L matrix are given next. Figure 3.11 is a phase space plot comparing the exact RKG solution and the approximation using the Putzer algorithm for the 20×20 truncation of the L matrix over 7 cycles of data. The corresponding differences in x and p are plotted in phase space in Figure 3.12 . The statistics for these figures are presented in Table 3.6. Figure 3.13 contains a phase plot of 2 cycles of data for this 20×20 case. Figure 3.14 presents a phase plot of the x and p differences for the 2 cycles of Figure 3.13.



PHASE PLOT exact & 12x12 apprx PUTZER

Figure 3.7



PHASE DIFF exact & 12x12 apprx PUTZER

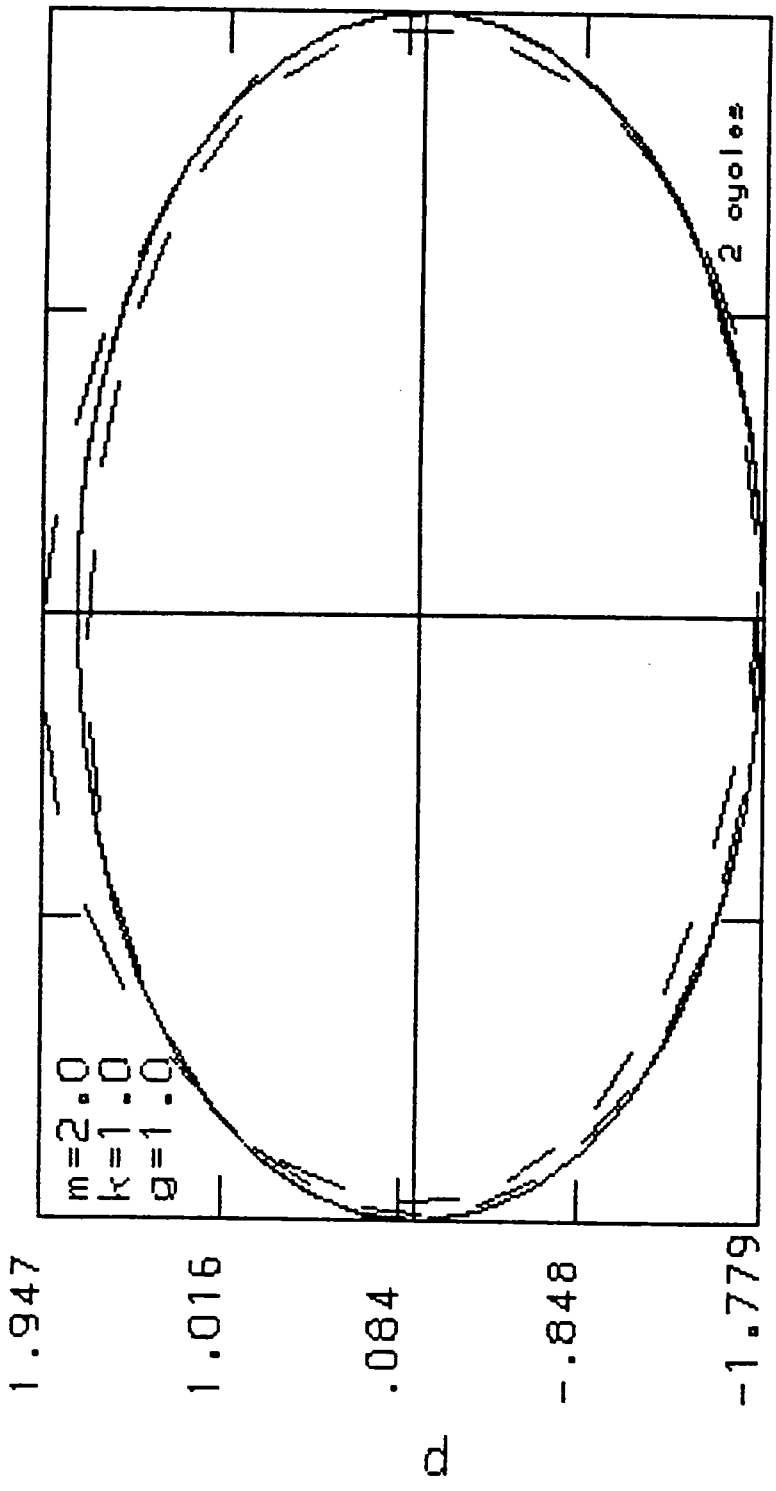
Figure 3.8

Table 3.5

Statistics for Δx and Δp for exact solution and 12x12 approximation using Putzer algorithm with RKG integration

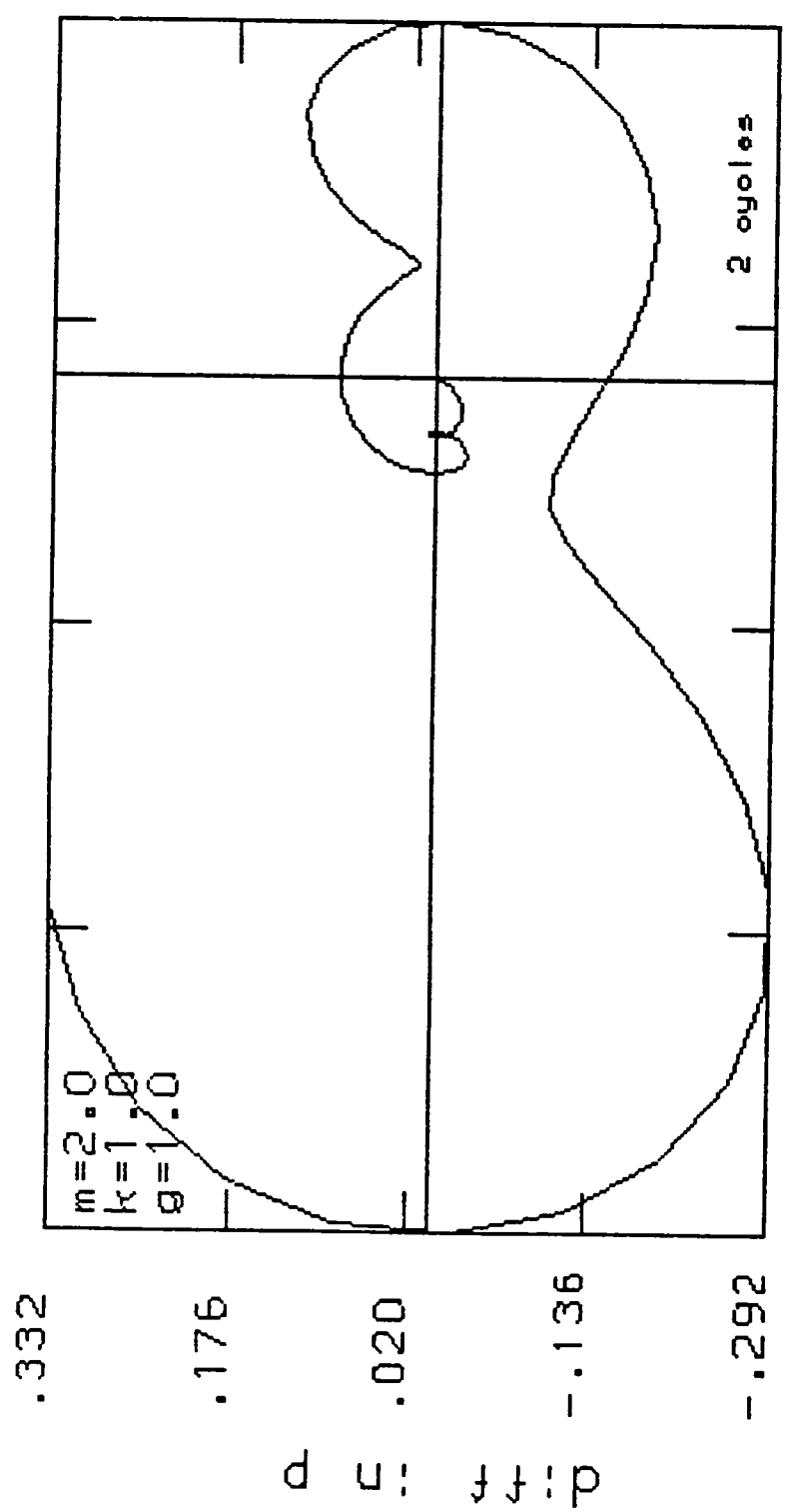
m = 2.0, k = 1.0, g = 1.0

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.013642	.032471
2	82	.087655	.116899
3	122	.238248	.324478
4	164	.496856	.752451
5	206	.898800	1.415691
6	250	1.530455	2.365497
7	294	2.364586	3.605183



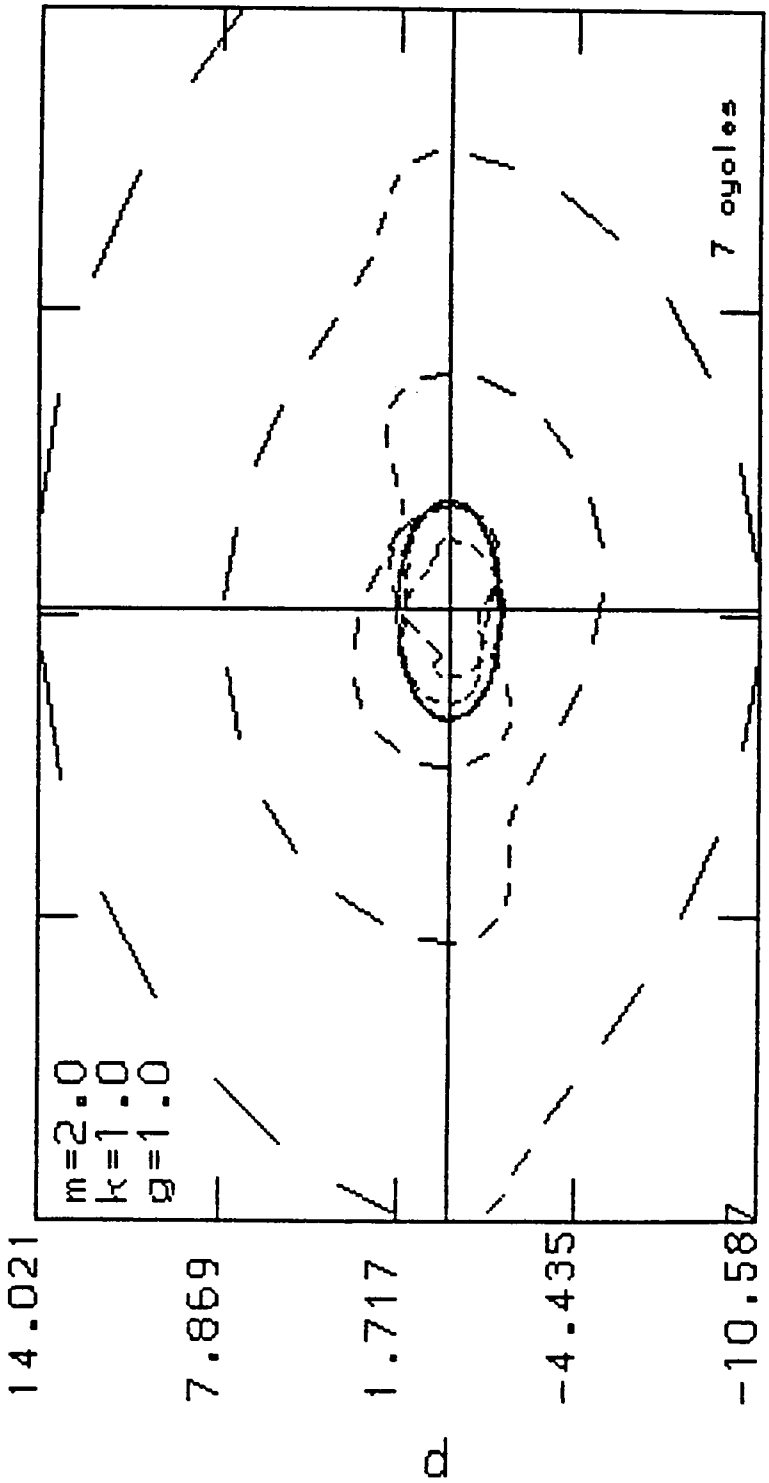
PHASE PLOT exact & 12x12 approx PUTZER

Figure 3.9



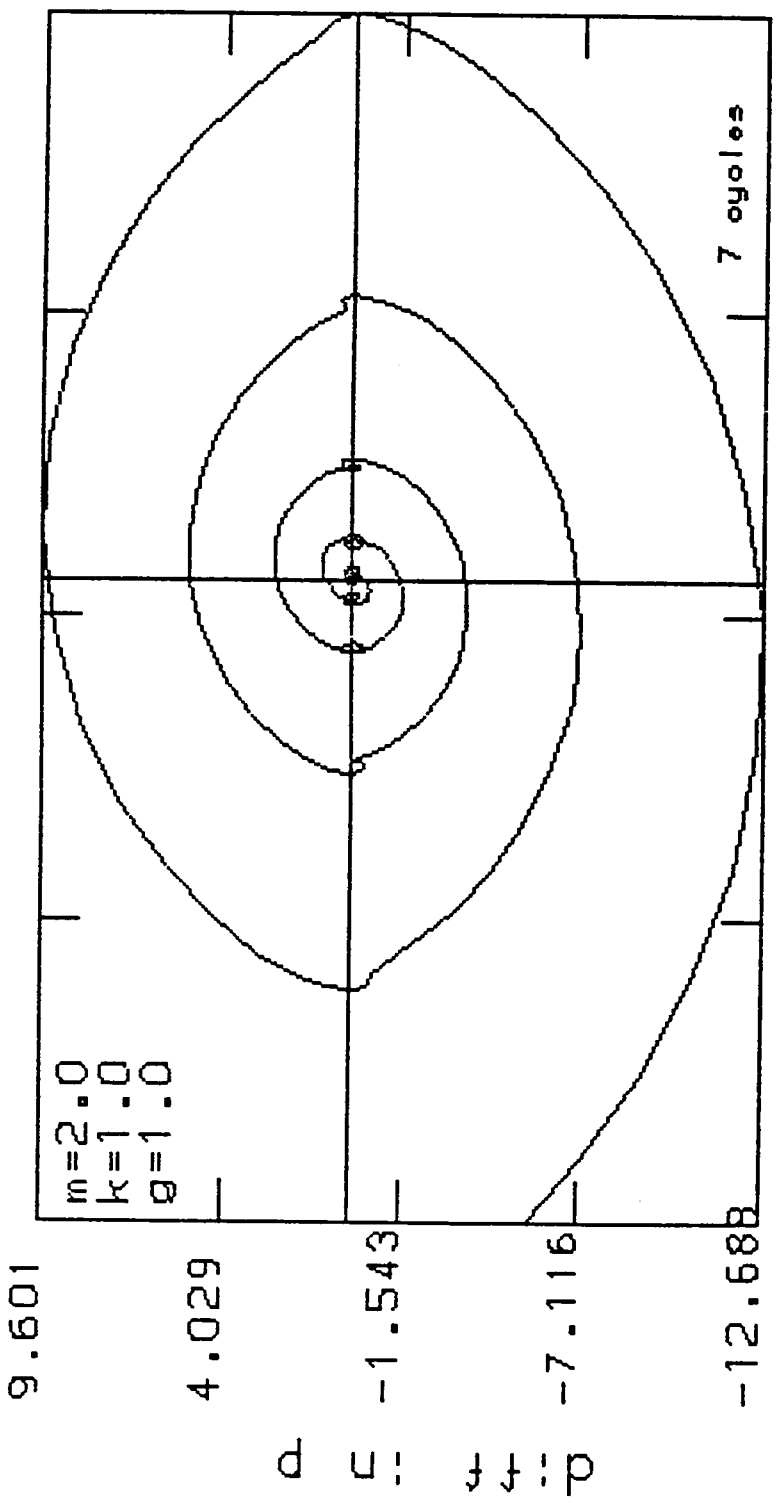
PHASE DIFF exact & 12x12 approx PUTZER

Figure 3.10



PHASE PLOT exact & 20x20 apprx PUTZER

Figure 3.11



PHASE DIFF exact & 20x20 approx PUTZER

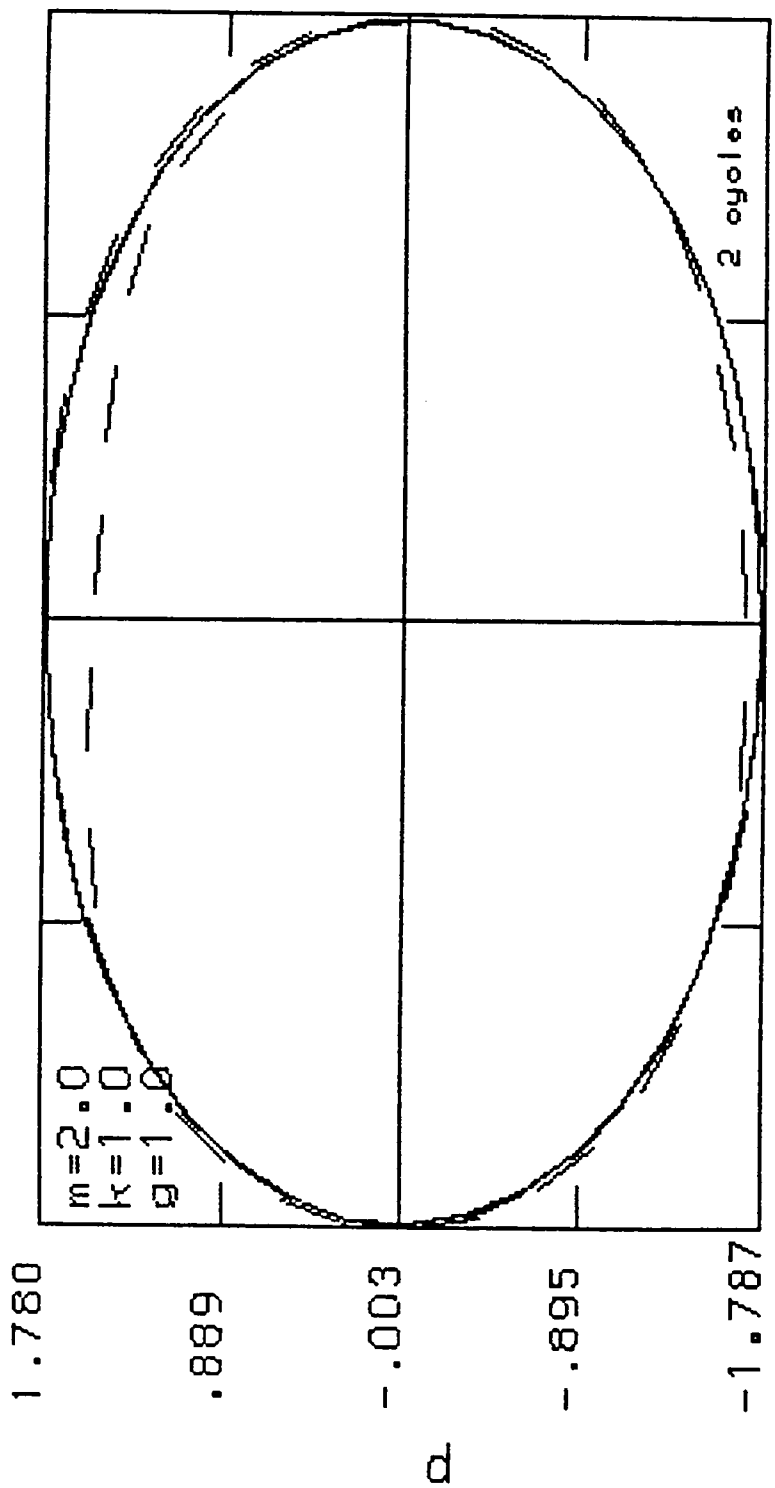
Figure 3.12

Table 3.6

Statistics for Δx and Δp for exact solution and 20x20 approximation using Putzer algorithm with RKG integration

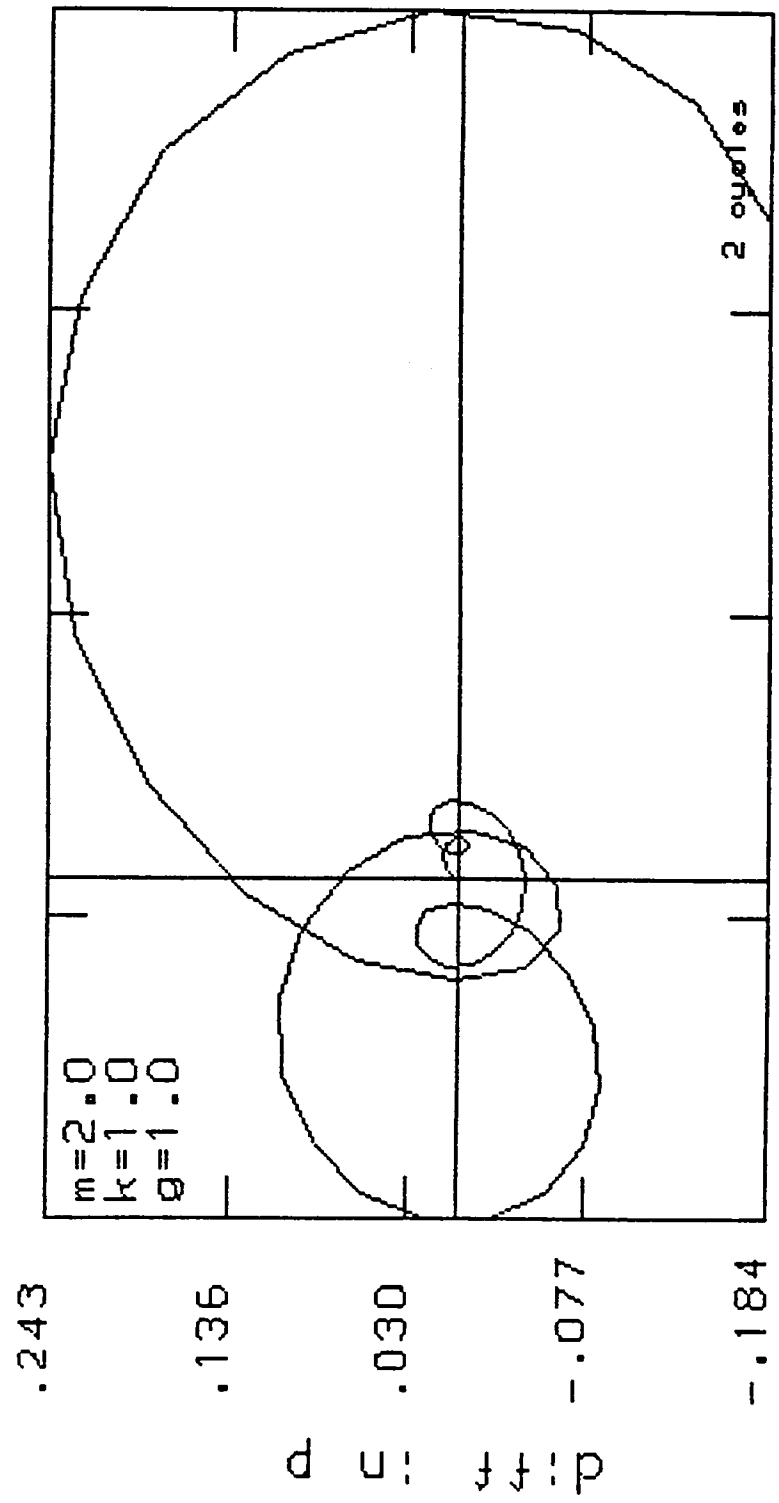
m = 2.0, k = 1.0, g = 1.0

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.003996	.014167
2	84	.034569	.069996
3	122	.080963	.216008
4	161	.253928	.372446
5	201	.575915	.820773
6	239	1.022416	1.661940
7	282	1.851547	3.003603



PHASE PLOT exact & 20x20 approx PUTZER

Figure 3.13



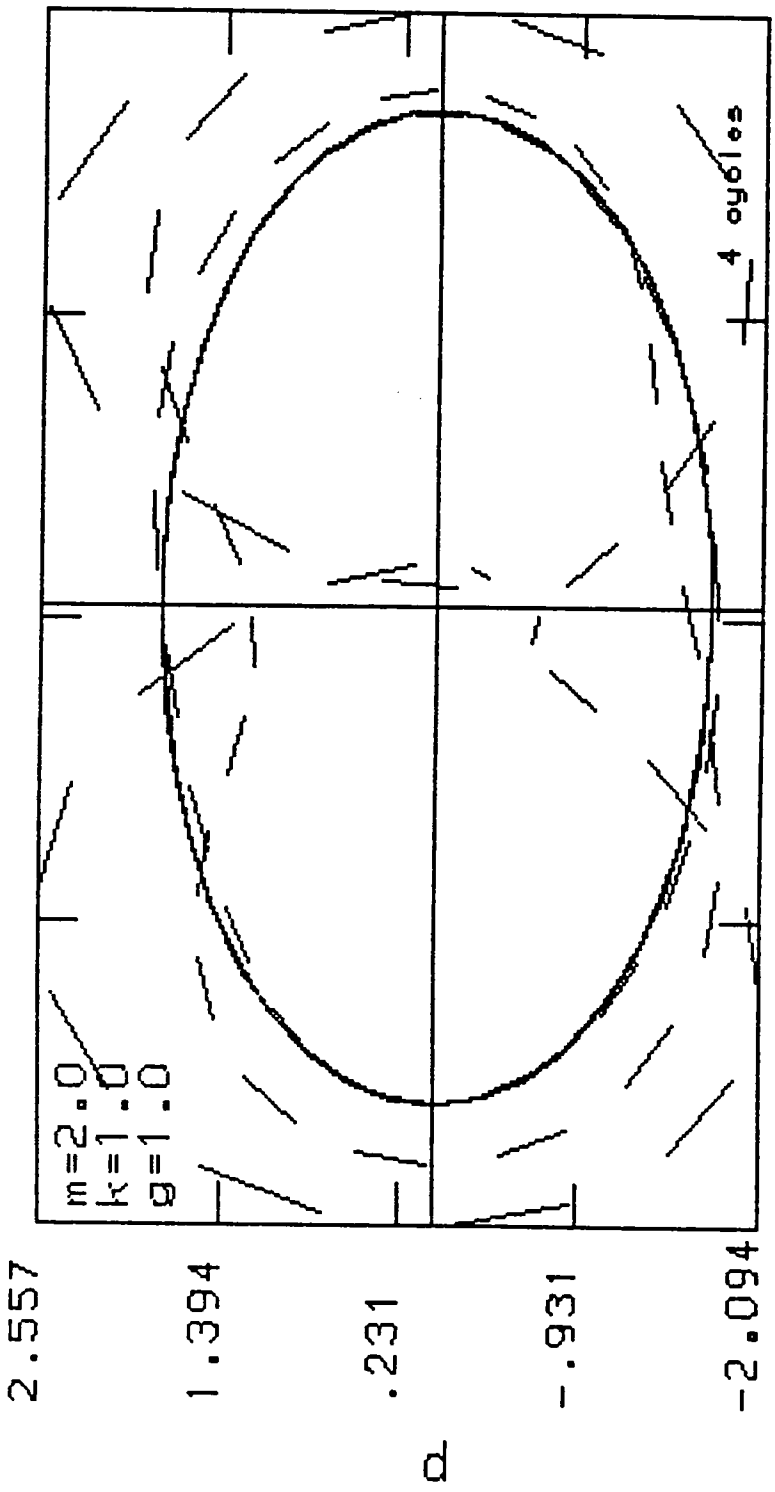
PHASE DIFF exact & 20x20 apprx PUTZER

Figure 3.14

Moving on to the data produced by the 30×30 truncation of the L matrix, Figure 3.15 presents a phase space plot comparing the exact RKG solution with the approximation using the Putzer algorithm. Differences in x and p between the exact and approximated solutions are presented in the phase plot of Figure 3.16 . Both figures cover 4 cycles of data. The statistics for these figures are presented in Table 3.7. Figure 3.17 shows the phase plots over 2 cycles comparing the exact RKG solution and approximated solutions for the 30×30 truncation. Figure 3.18 presents a plot in phase space of the x and p differences corresponding to Figure 3.17.

Next, the position and momentum produced by the approximation using the Putzer algorithm applied to the 42×42 truncation of the L matrix is compared to the exact RKG position and momentum via Figure 3.19, which presents a phase space plot of the two coordinates over one cycle of data. The corresponding differences in x and p between the exact and approximated solutions are presented in Figure 3.20. The statistics for these figures are presented in Table 3.8.

No data were produced for the 56×56 truncation of the L matrix since the program was not small enough to be loaded in the available memory of the personal computer being used to conduct the investigation.



PHASE PLOT exact & 30x30 apprx PUTZER

Figure 3.15

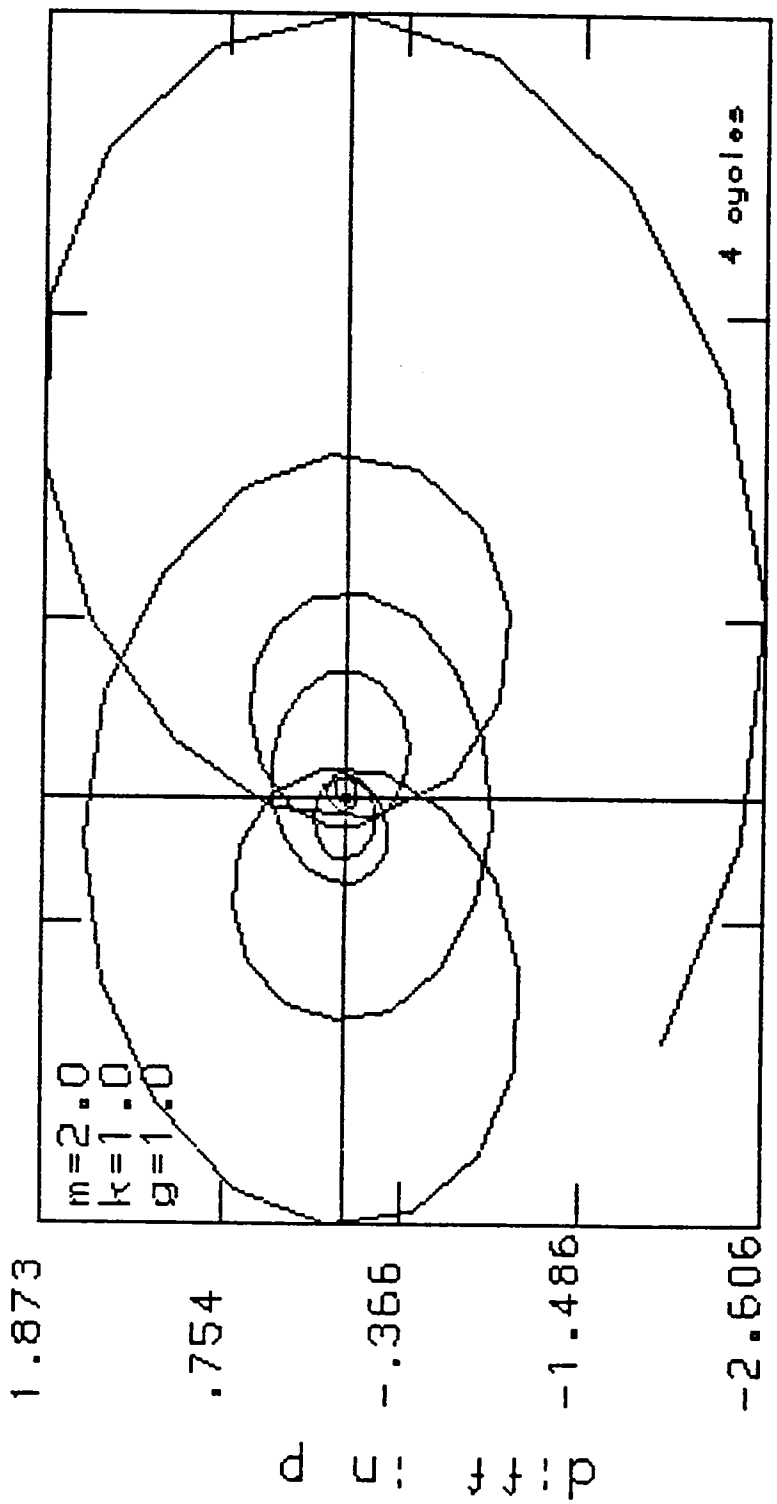

PHASE DIFF exact & 30x30 apprx PUTZER
m=2.0
k=1.0
g=1.0
4 cycles
1.873
.754
-.366
-1.486
-2.606
diff in p
-.477
-.139
.200
.539
.878
diff in x

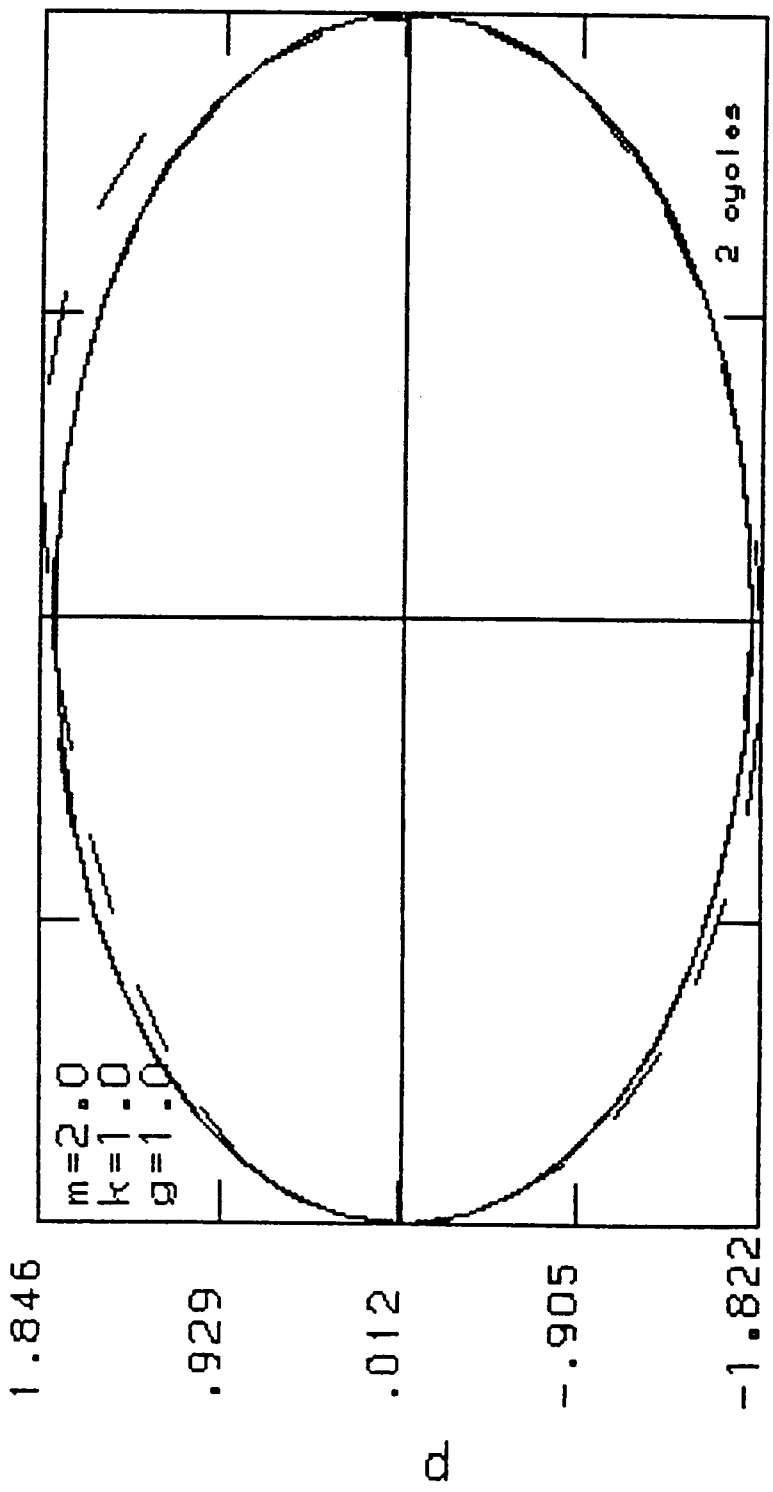
Figure 3.16

Table 3.7

Statistics for Δx and Δp for exact solution and 30x30 approximation using Putzer algorithm with RKG integration

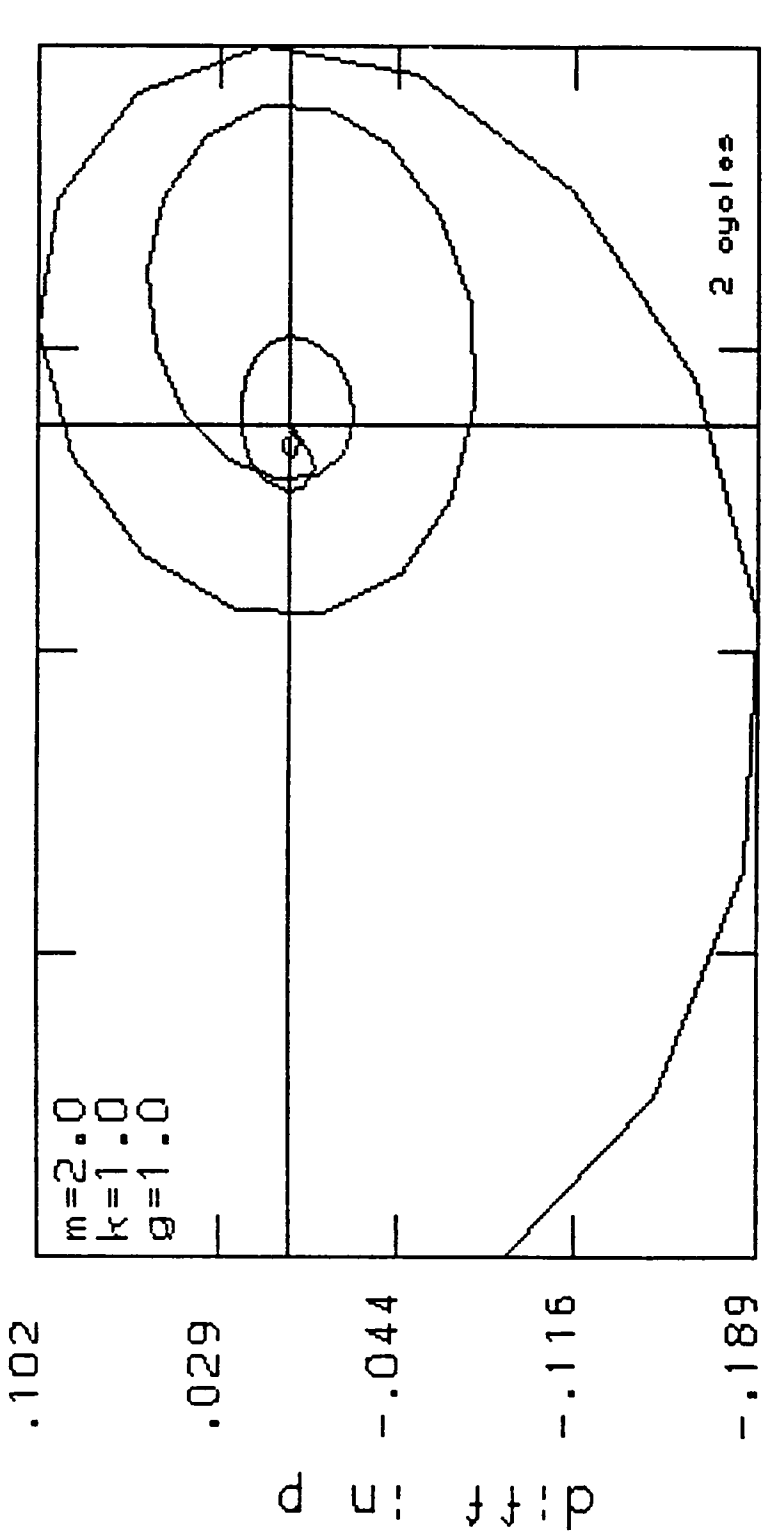
m = 2.0, k = 1.0, g = 1.0

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.001427	.005493
2	83	.013289	.050820
3	126	.056483	.223270
4	171	.191608	.628312



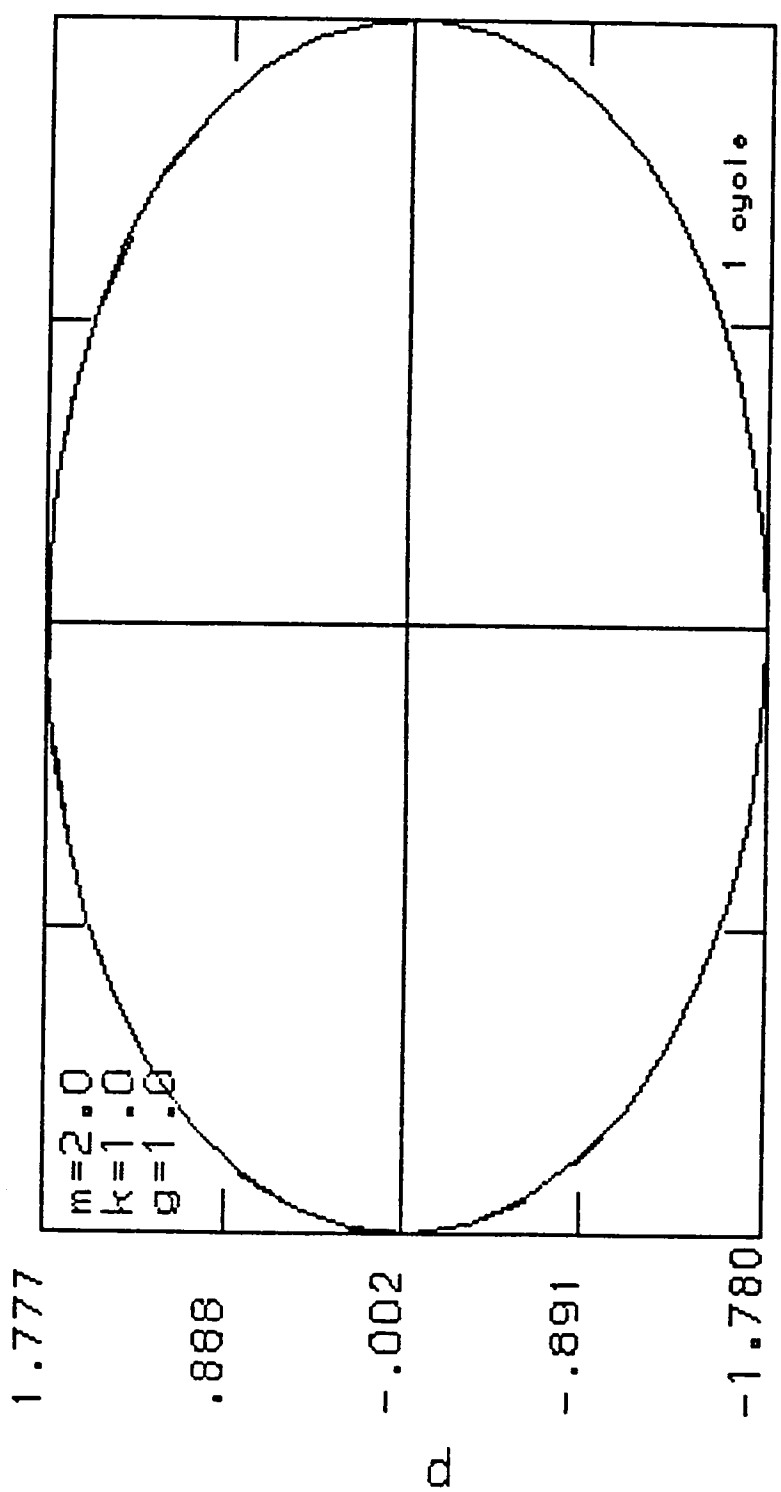
PHASE PLOT exact & 30x30 apprx PUTZER

Figure 3.17



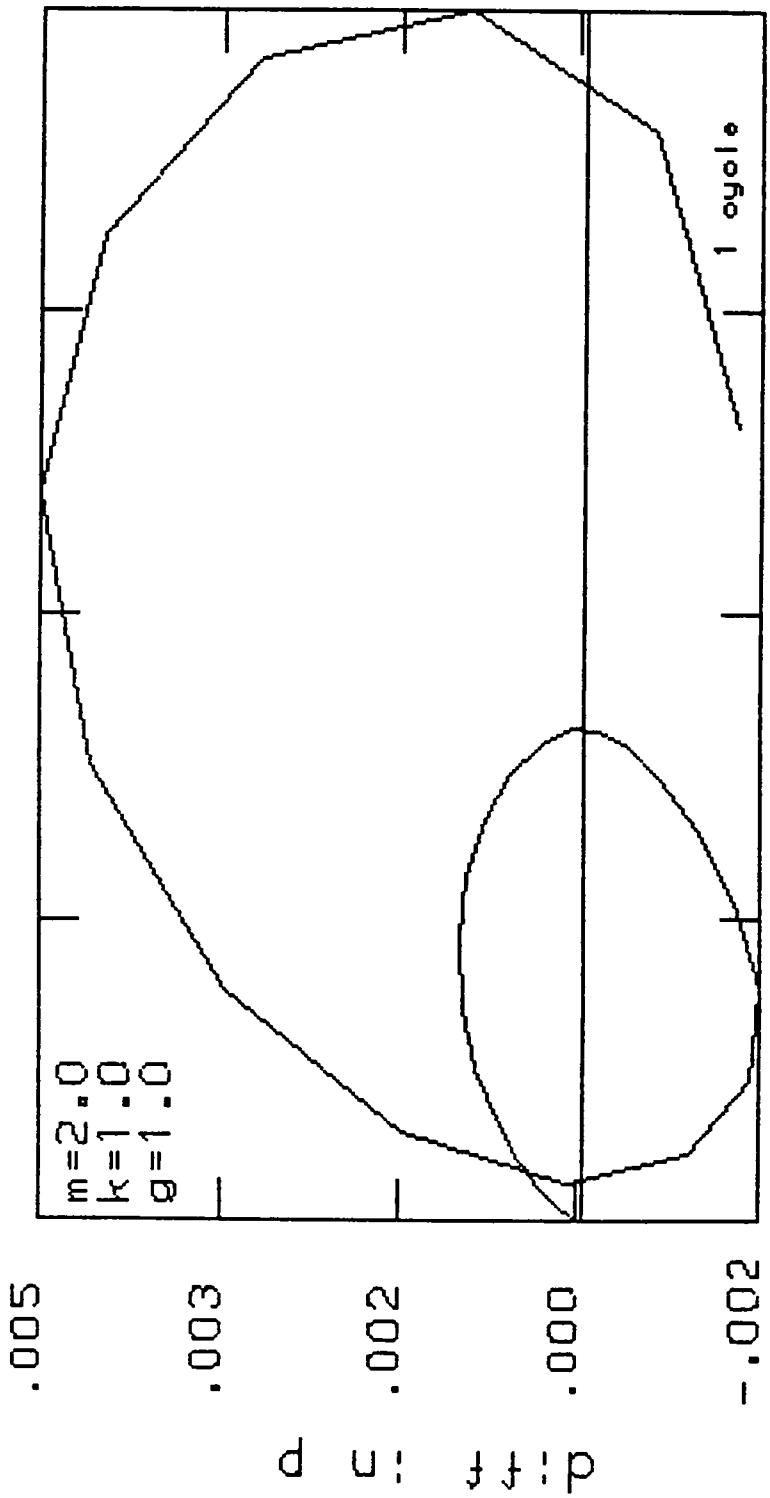
PHASE DIFF exact & 30x30 apprx PUTZER

Figure 3.18



PHASE PLOT exact & 42x42 apprx PUTZER

Figure 3.19



PHASE DIFF exact & 42x42 apprx PUTZER

Figure 3.20

Table 3.8

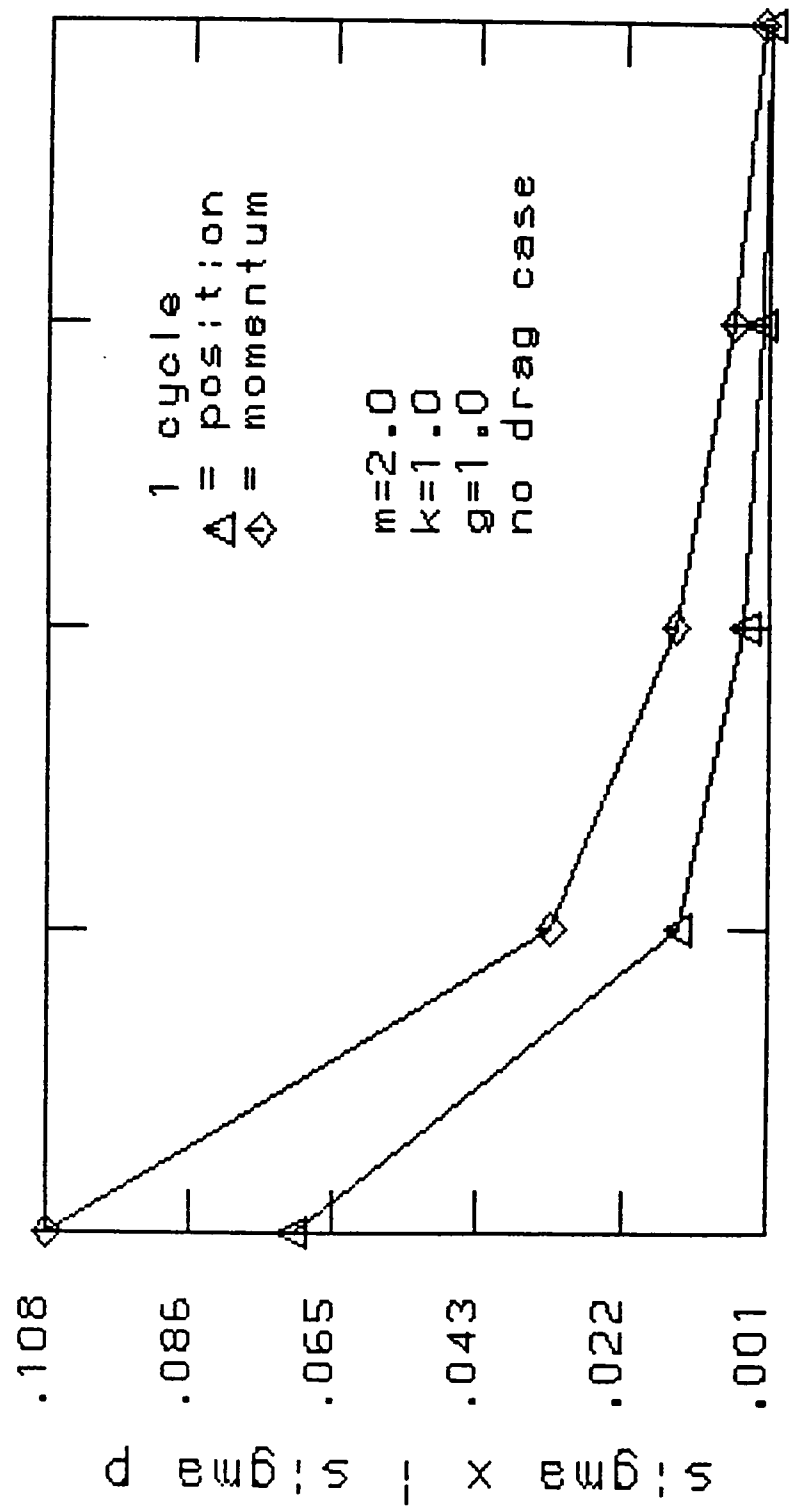
Statistics for Δx and Δp for exact solution and 42x42 approximation using Putzer algorithm with RKG integration

m = 2.0, k = 1.0, g = 1.0

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.000587	.001510

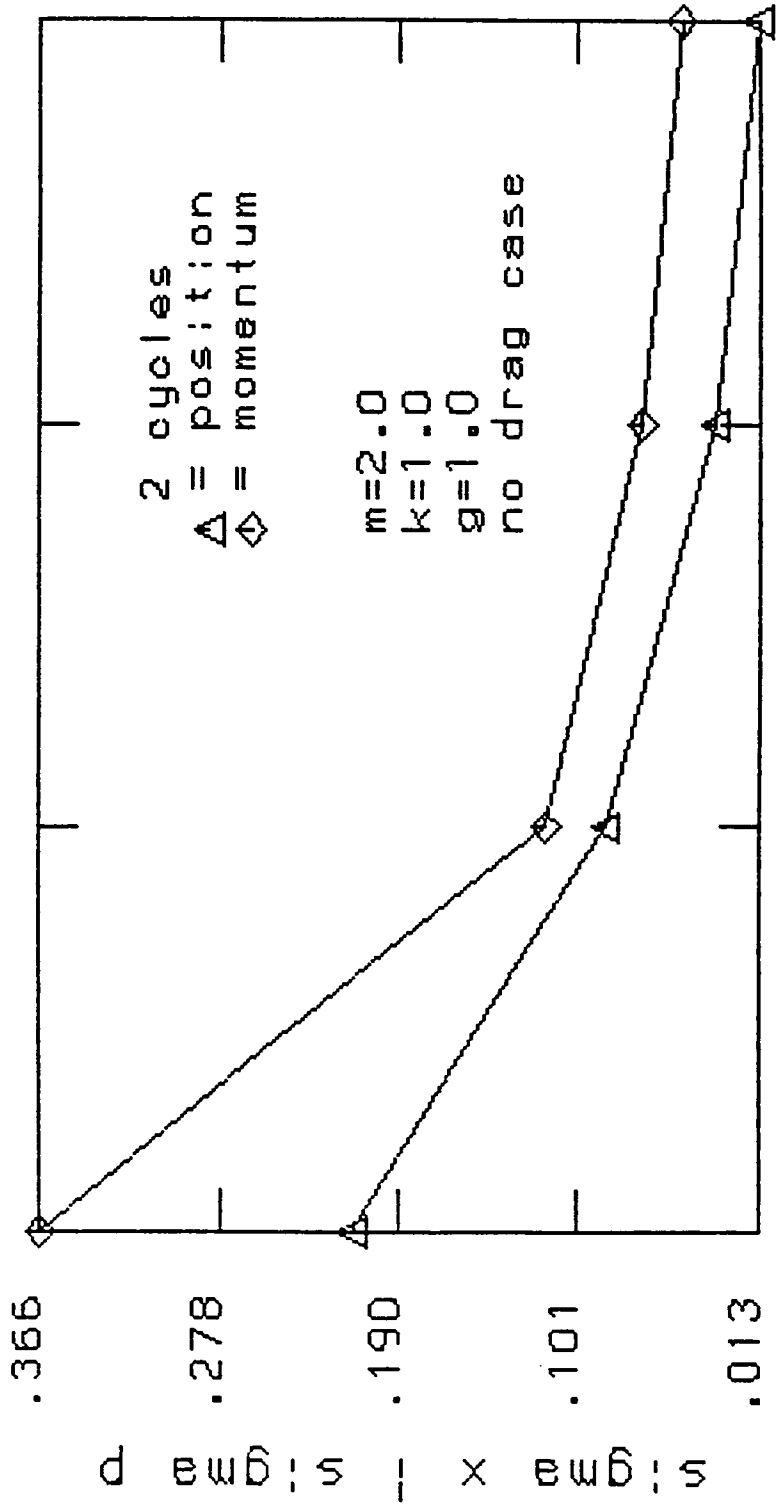
A summary of how the statistics for x and p changed when higher and higher truncations of the L matrix were made is graphically presented in the series of Figures 3.21 through 3.24, covering data for 1 cycle, 2 cycles, 3 cycles, and 4 cycles respectively. Data were not obtainable for cycles 2 through 4 from the 42×42 truncation. Considering all 4 cycles of data, the standard deviation for the position is monotonically decreasing as higher truncations are utilized in the approximation scheme. The standard deviation for the momentum is monotonically decreasing for 1 cycle and 2 cycle data as higher truncations are implemented into the approximation scheme. However, for 3 cycle and 4 cycle data, the standard deviation for the momentum is monotonically decreasing only to the 20×20 truncation of the L matrix. This reflects the deterioration of the data mentioned previously. The data still contain secular components that have not been removed numerically. Perhaps these secular components are coming into play to a greater degree when the 30×30 and higher truncations are implemented.

As a means by which to check if the results produced by the personal computer were not jeopardized due to the computer's size, both the code to produce the eigenvalues and to implement the approximation scheme utilizing the Putzer algorithm for the 20×20 truncation of the L matrix



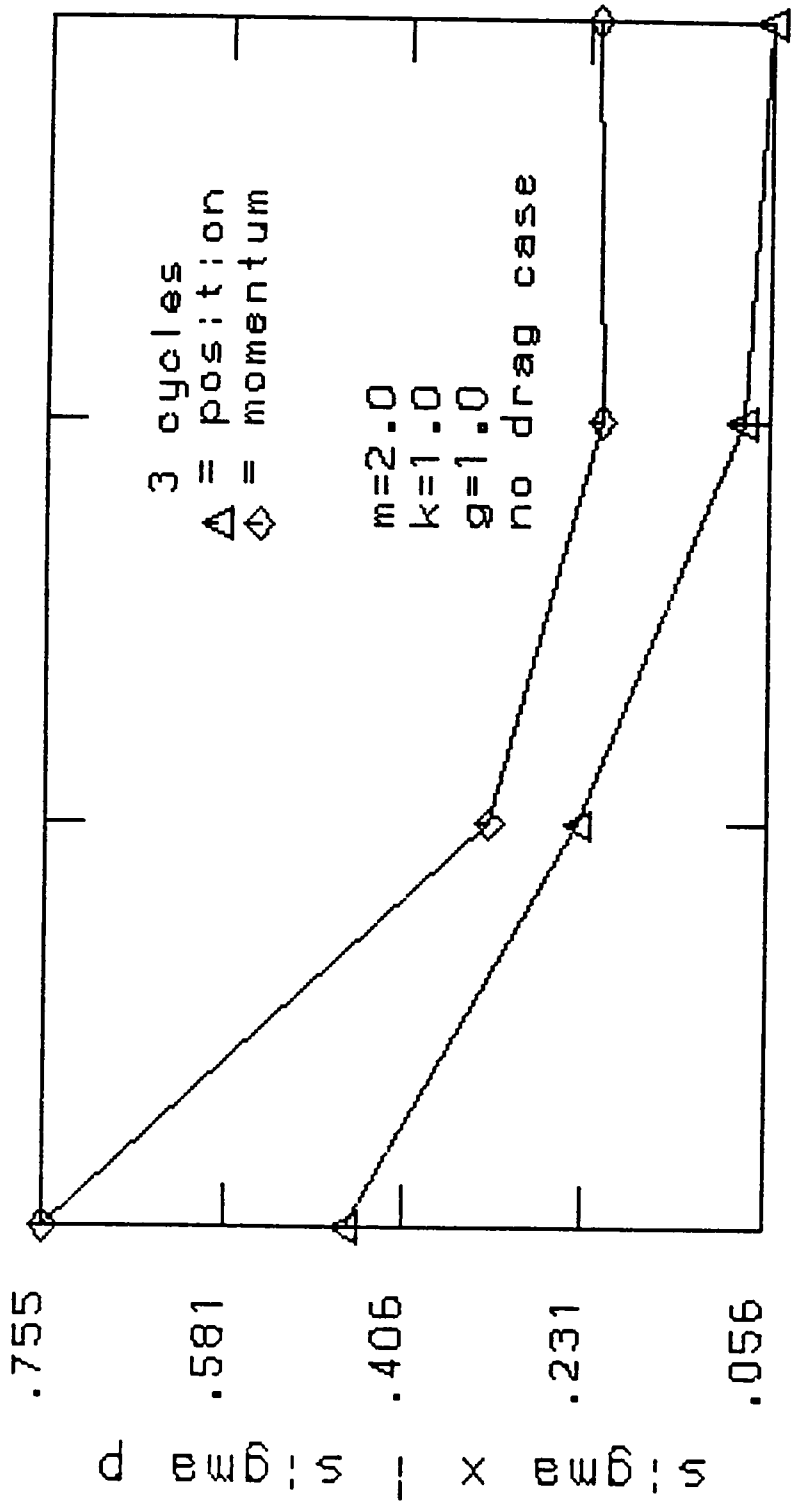
STATISTICS by Trunc Size for 1 Cycle

Figure 3.21



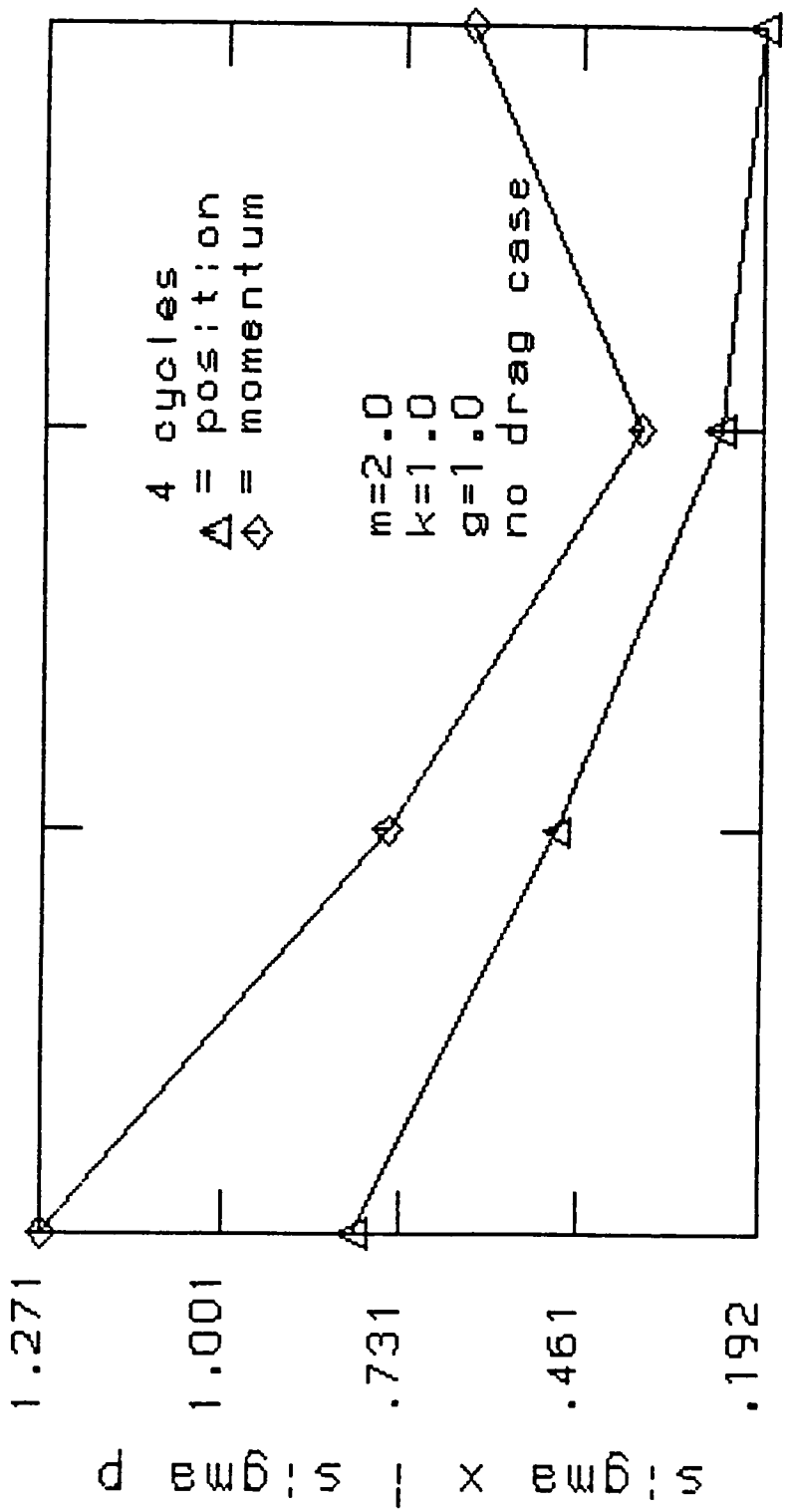
STATISTICS by Trunc Size for 2 Cycles

Figure 3.22



STATISTICS by Trunc Size for 3 Cycles

Figure 3.23



STATISTICS by Trunc Size for 4 Cycles

Figure 3.24

were converted to run on a MicroVAX. A statistical and graphical comparison was made between the data from the personal computer and the resultant data of the MicroVAX. Both sets of data produced the same graph when plotted in phase space. A one-sigma value for the dfferences in the position was found to be .000340 when 282 data points were analyzed. A one-sigma value for the differences in the momentum was found to be .001481, again when 282 data points were analyzed. These facts would tend to indicate that the limited wordsize and limited memory of the personal computer were not a factor in the deterioration of the data. Rather these facts would tend to indicate that there is some subtle error or errors in the code that was used to produce data for the various truncations of the L matrix.

In the next chapter the Hamiltonian studied in this chapter will be modified to include a term representing the effects of a drag force on the perturbed harmonic oscillator. The Liouville operator will be used as before but with a modification of one of the derivatives involved. A matrix will be produced and data generated for various truncations of the L matrix, and the data will be analyzed in a manner similar to that performed in this chapter.

CHAPTER 4

STUDY OF APPROXIMATION TECHNIQUE USING PUTZER ALGORITHM ON SIMPLE HARMONIC OSCILLATOR WITH X^4 PERTURBATION AND DRAG LINEAR IN THE VELOCITY

In order to ascertain how well the approximation schema under consideration performs when a non-conservative or a dissipative force such as a drag force is acting, the Hamiltonian,

$$H = \frac{p^2}{2m} + \frac{1}{2} kx^2 + \frac{g}{4!} x^4$$

is again considered, together with a drag force having the general form

$$f = -\alpha(\dot{x})^d ,$$

where, for simplicity, the integer d has been taken to be 1 producing a drag force which is linearly dependent on the velocity. Since d = 1, this f is the result of the derivative of the Rayleigh dissipation function F

$$F = -\frac{1}{2} \alpha (\dot{x})^2 .$$

The Lagrangian for this case is

$$L = \frac{1}{2} mv^2 - \frac{1}{2} kx^2 - \frac{g}{4!} x^4 .$$

Lagrange's equations for the dissipative situation are given by

$$\frac{d}{dt}\left\{\frac{\partial L}{\partial \dot{q}_j}\right\} - \frac{\partial L}{\partial q_j} + \frac{\partial F}{\partial \dot{q}_j} = 0$$

and yield the second order non-linear differential equation

$$\ddot{x} - \frac{\alpha}{m}\dot{x} + \frac{k}{m}x + \frac{g}{3!\,m}x^3 = 0 .$$

Each term of this equation can be multiplied by $\dot{x}$, with most of the resultant terms rewritten using differentials to produce

$$\frac{d}{dt}\left\{(\ddot{x})^2 + \frac{k}{m}x^2 + \frac{2\,g}{4!\,m}x^4\right\} - \frac{2\alpha}{m}(\dot{x})^2 = 0 .$$

Here the $\dot{x}$ term presents trouble when trying to find an exact differential in order that a first quadrature may be completed. An attempt to solve the original differential equation can be made by making the following substitutions

$$\dot{x} = z ,$$
$$\ddot{x} = z\frac{dz}{dx} ,$$

and thereby transforming the differential equation into

$$z\frac{dz}{dx} - \frac{\alpha}{m}z + \frac{k}{m}x + \frac{g}{3!\,m}x^3 = 0 .$$

This equation has the form

$$zz' + f(x) = g(x)z$$

of Abel's equation of the second kind. An integrating

factor can be found only if there exists an n such that

$$(n-1)\,g(x) = n^2\left[\frac{f}{g}\right]' ,$$

then

$$\phi(x) \equiv -n\,\frac{f}{g}$$

and the integrating factor is given by (Murphy 1960: 25-26 & 255)

$$I(x,z) = (\phi + z)^{-n} .$$

Such an n does not exist for this differential equation.

The Liouville operator, $\mathcal{L}$, will be defined in the same manner as before, and then adjusted, e.g.

$$\mathcal{L}(\xi) = [\xi, H] = \frac{\partial \xi}{\partial x}\frac{\partial H}{\partial p} - \frac{\partial \xi}{\partial p}\frac{\partial H'}{\partial x}$$

where, for consistency, the following expression

$$\frac{\partial H'}{\partial x} = \frac{\partial H}{\partial x} + \frac{\alpha}{m}\,p$$

must be used. This replacement in the definition of the Liouville operator produces the same results as if one had simply differentiated the function $\xi(x,p)$ with respect to t and substituted the following expression for $\dot{p}$,

$$\dot{p} = -kx - \frac{g}{3!}\,x^3 - \frac{\alpha}{m}\,p .$$

A simple calculation will bear this out. Either method, whether employing the adjusted Liouville operator and making the necessary adjustment in the partial of H with respect to x, or simply taking the time derivative and using the above expression to replace $\dot{p}$ will lead to the same L matrix. A monomial basis similar to the one previously obtained is chosen, but with terms from all possible integer powers, ℓ, of the expansion of $(x + p)^{\ell}$. In the case without drag, only odd integer powers appeared. For the purpose of the study here, the maximum value of ℓ will be limited to 9, thereby producing a 54×54 matrix. The L matrix constructed from these basis vectors for the 54×54 truncation can be partitioned into the following block form

$$L = \begin{bmatrix} \mathbf{A} & 0 & A & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{B} & 0 & B & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{C} & 0 & C & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{D} & 0 & D & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{E} & 0 & E & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{F} & 0 & F & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{G} & 0 & G \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{H} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{J} \end{bmatrix},$$

where the boldface entries are blocks of various dimensions. Those blocks along the main diagonal are matrices whose elements are dependent upon k, m, and α, while those non-zero blocks appearing on the super diagonal

are matrices whose elements are dependent only on g. The various non-zero blocks are defined as follows:

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{1}{m} \\ -k & -\frac{\alpha}{m} \end{bmatrix}, \quad \mathcal{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\frac{g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 0 & \frac{2}{m} & 0 \\ -k & -\frac{\alpha}{m} & \frac{1}{m} \\ 0 & -2k & -2\frac{\alpha}{m} \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -\frac{g}{3!} & 0 & 0 & 0 & 0 \\ 0 & -\frac{2g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} 0 & \frac{3}{m} & 0 & 0 \\ -k & -\frac{\alpha}{m} & \frac{2}{m} & 0 \\ 0 & -2k & -2\frac{\alpha}{m} & \frac{1}{m} \\ 0 & 0 & -3k & -3\frac{\alpha}{m} \end{bmatrix}, \quad \mathcal{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{g}{3!} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2g}{3!} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{D} = \begin{bmatrix} 0 & \frac{4}{m} & 0 & 0 & 0 \\ -k & -\frac{\alpha}{m} & \frac{3}{m} & 0 & 0 \\ 0 & -2k & -2\frac{\alpha}{m} & \frac{2}{m} & 0 \\ 0 & 0 & -3k & -3\frac{\alpha}{m} & \frac{1}{m} \\ 0 & 0 & 0 & -4k & -\frac{\alpha}{m} \end{bmatrix},$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{g}{31} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2g}{3!} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3g}{3!} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{4g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{E} = \begin{bmatrix} 0 & \frac{5}{m} & 0 & 0 & 0 & 0 \\ -k & -\frac{\alpha}{m} & \frac{4}{m} & 0 & 0 & 0 \\ 0 & -2k & -2\frac{\alpha}{m} & \frac{3}{m} & 0 & 0 \\ 0 & 0 & -3k & -3\frac{\alpha}{m} & \frac{2}{m} & 0 \\ 0 & 0 & 0 & -4k & -4\frac{\alpha}{m} & \frac{1}{m} \\ 0 & 0 & 0 & 0 & -5k & -5\frac{\alpha}{m} \end{bmatrix},$$

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3g}{3!} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{4g}{3!} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{5g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$F = \begin{bmatrix} 0 & \frac{6}{m} & 0 & 0 & 0 & 0 & 0 \\ -k & -\frac{\alpha}{m} & \frac{5}{m} & 0 & 0 & 0 & 0 \\ 0 & -2k & -2\frac{\alpha}{m} & \frac{4}{m} & 0 & 0 & 0 \\ 0 & 0 & -3k & -3\frac{\alpha}{m} & \frac{3}{m} & 0 & 0 \\ 0 & 0 & 0 & -4k & -4\frac{\alpha}{m} & \frac{2}{m} & 0 \\ 0 & 0 & 0 & 0 & -5k & -5\frac{\alpha}{m} & \frac{1}{m} \\ 0 & 0 & 0 & 0 & 0 & -6k & -6\frac{\alpha}{m} \end{bmatrix},$$

$$\mathcal{F} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{4g}{3!} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{5g}{3!} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{6g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$G = \begin{bmatrix} 0 & \frac{7}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\ -k & -\frac{\alpha}{m} & \frac{6}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & -2k & -2\frac{\alpha}{m} & \frac{5}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & -3k & -3\frac{\alpha}{m} & \frac{4}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & -4k & -4\frac{\alpha}{m} & \frac{3}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & -5k & -5\frac{\alpha}{m} & \frac{2}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & -6k & -6\frac{\alpha}{m} & \frac{1}{m} \\ 0 & 0 & 0 & 0 & 0 & 0 & -7k & -7\frac{\alpha}{m} \end{bmatrix},$$

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{4g}{3!} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{5g}{3!} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{6g}{3!} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{7g}{3!} & 0 & 0 & 0 \end{bmatrix},$$

$$
H = \begin{bmatrix}
0 & \frac{8}{m} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-k & -\frac{\alpha}{m} & \frac{7}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -2k & -2\frac{\alpha}{m} & \frac{6}{m} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -3k & -3\frac{\alpha}{m} & \frac{5}{m} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -4k & -4\frac{\alpha}{m} & \frac{4}{m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -5k & -5\frac{\alpha}{m} & \frac{3}{m} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -6k & -6\frac{\alpha}{m} & \frac{2}{m} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -7k & -7\frac{\alpha}{m} & \frac{1}{m} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -8k & -8\frac{\alpha}{m}
\end{bmatrix},
$$

$$J = \begin{bmatrix} 0 & \frac{9}{m} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k & -\frac{\alpha}{m} & \frac{8}{m} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2k & -2\frac{\alpha}{m} & \frac{7}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3k & -3\frac{\alpha}{m} & \frac{6}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4k & -4\frac{\alpha}{m} & \frac{5}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5k & -5\frac{\alpha}{m} & \frac{4}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6k & -6\frac{\alpha}{m} & \frac{3}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -7k & -7\frac{\alpha}{m} & \frac{2}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -8k & -8\frac{\alpha}{m} & \frac{1}{m} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9k & -9\frac{\alpha}{m} \end{bmatrix}.$$

The L matrix associated with each truncation defined by the various values of ℓ from 1 to 9 is an upper Hessenberg matrix. The drag coefficient has been introduced only along the main diagonal. For values of ℓ greater than 1 the terms containing the drag coefficient are contained in those non-zero blocks appearing on the super diagonals above the main diagonal of blocks. The super diagonal chosen to contain blocks whose elements are dependent on the drag coefficient as a factor is

determined by the power of the velocity taken on the drag force. A power of 1 produces terms along the main diagonal, a power of 2 produces terms in the block super diagonal immediately above the main block diagonal, while a power of 3 produces drag coefficient entries in the block super diagonal located two block diagonals above the main block diagonal. This pattern is continued for any size ℓ. These L matrix variations will not be discussed in detail here, however, as this study will be confined to that L matrix produced for the drag force that is linearly dependent upon the velocity (d=1).

An examination of the eigenvalue structure of the 54×54 truncation of the L matrix is now conducted. The matrix is partitioned in accordance with previously mentioned procedures. The eigenvalues are generally characterized by either of three forms. The first form is characterized by a negative real part and a positive imaginary part. The second form is the complex conjugate of the first form. The third form has a negative real part and zero complex part, and occurs, usually only once, in those partitions having an odd number of basis elements. For fixed values of k and g, and for varying values of α, the drag coefficient, the eigenvalue spectrum, denoted by $\sigma(\lambda)$, has the structural form shown in Table 4.1 below for the 54×54 truncation of the L matrix. Notationally, let

Table 4.1

Eigenvalue Spectrum by Partition of L Matrix for Perturbed SHO with Drag

$m = 2.0 \qquad k = 1.0 \qquad g = 1.0$

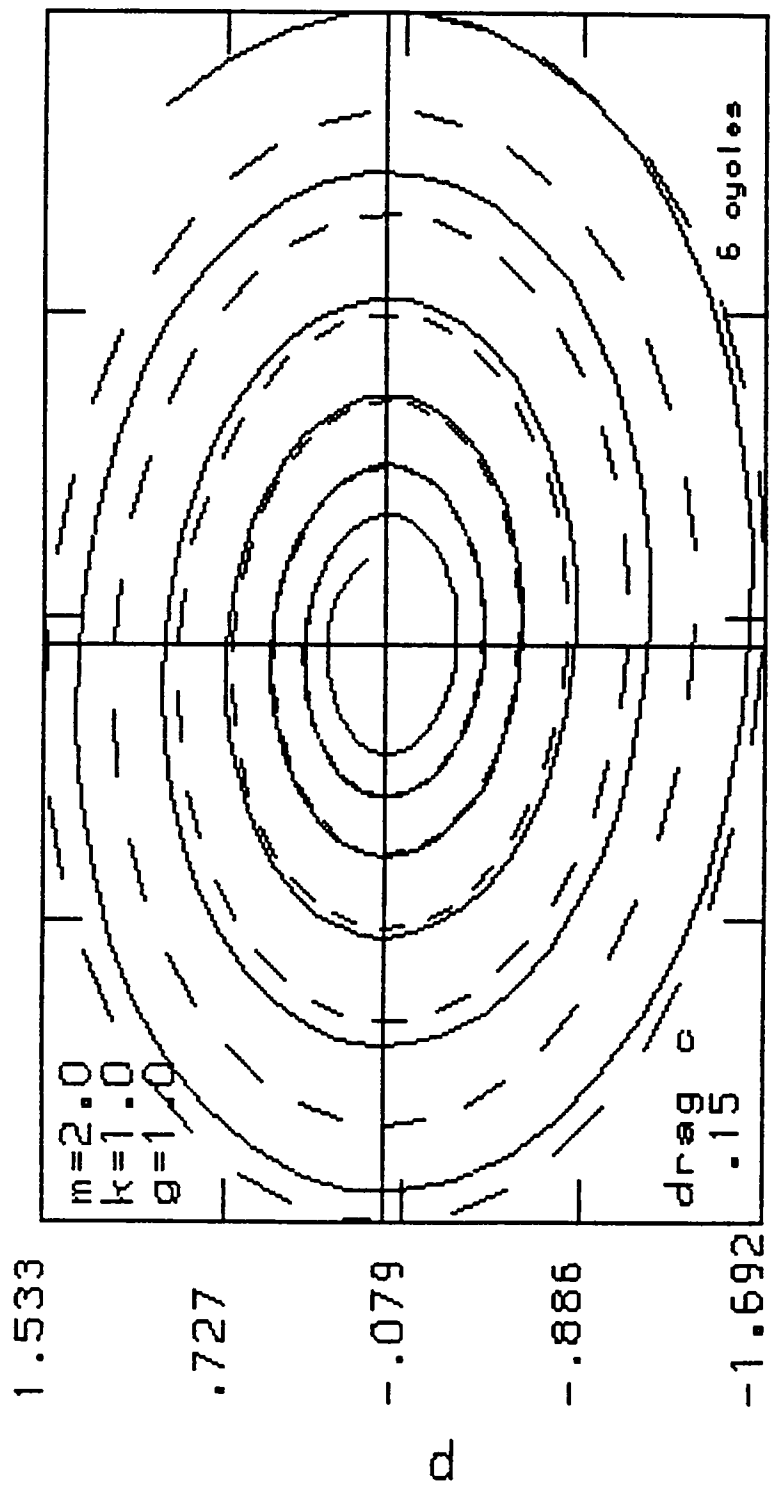
$$\alpha = .15 \qquad \beta = \sqrt{\left[\frac{\alpha}{m}\right]^2 - 4\,\frac{k}{m}} = .706111$$

partition number	eigenvalue	
	real part	imag part
1	$-1(\alpha/m)/2$	$\pm 1\beta$
2	$-2(\alpha/m)/2$	$\pm 2\beta$ 0
3	$-3(\alpha/m)/2$	$\pm 3\beta$ $\pm\beta$
4	$-4(\alpha/m)/2$	$\pm 4\beta$ $\pm 2\beta$ 0
5	$-5(\alpha/m)/2$	$\pm 5\beta$ $\pm 3\beta$ $\pm\beta$
6	$-6(\alpha/m)/2$	$\pm 6\beta$ $\pm 4\beta$ $\pm 2\beta$ 0
7	$-7(\alpha/m)/2$	$\pm 7\beta$ $\pm 5\beta$ $\pm 3\beta$ $\pm\beta$
8	$-8(\alpha/m)/2$	$\pm 8\beta$ $\pm 6\beta$ $\pm 4\beta$ $\pm 2\beta$ 0
9	$-9(\alpha/m)/2$	$\pm 9\beta$ $\pm 7\beta$ $\pm 5\beta$ $\pm 3\beta$ $\pm\beta$

β denote the smallest magnitude of the imaginary part of the eigenvalue that has the lowest overall magnitude. This structure of eigenvalues is constant regardless of the value of the drag constant chosen. The form of the eigenvalue structure is related to the partitions of the L matrix. The partitions take their form from the fact that monomials were used for a basis of independent vectors.

The approximation scheme incorporating the Putzer algorithm was utilized to program the 9×9 truncation, the 14×14 truncation, and the 20×20 truncation of the L matrix. The program for each truncation was used to produce data for the drag coefficients of .15, .30, .45, and .60, for the times of 21.0, 30.0, and 60.0 sec. Exact RKG solution data were obtained for each time frame. Only data for the 60.0 sec time frame using the drag coefficient of .15 are presented here. The complete complement of the data generated will not be shown due to its similar nature, however a representative sample will be shown for the 9×9, 14×14, and 20×20 truncations.

For the 9×9 truncation a comparative phase plot of x and p produced by both the exact RKG solution (solid line) and the approximation using the Putzer algorithm (dashed line) is presented in Figure 4.1 for 6 cycles of data with a drag coefficient of .15 . In contrast to the non-drag case, the motion is spirally inward indicating the

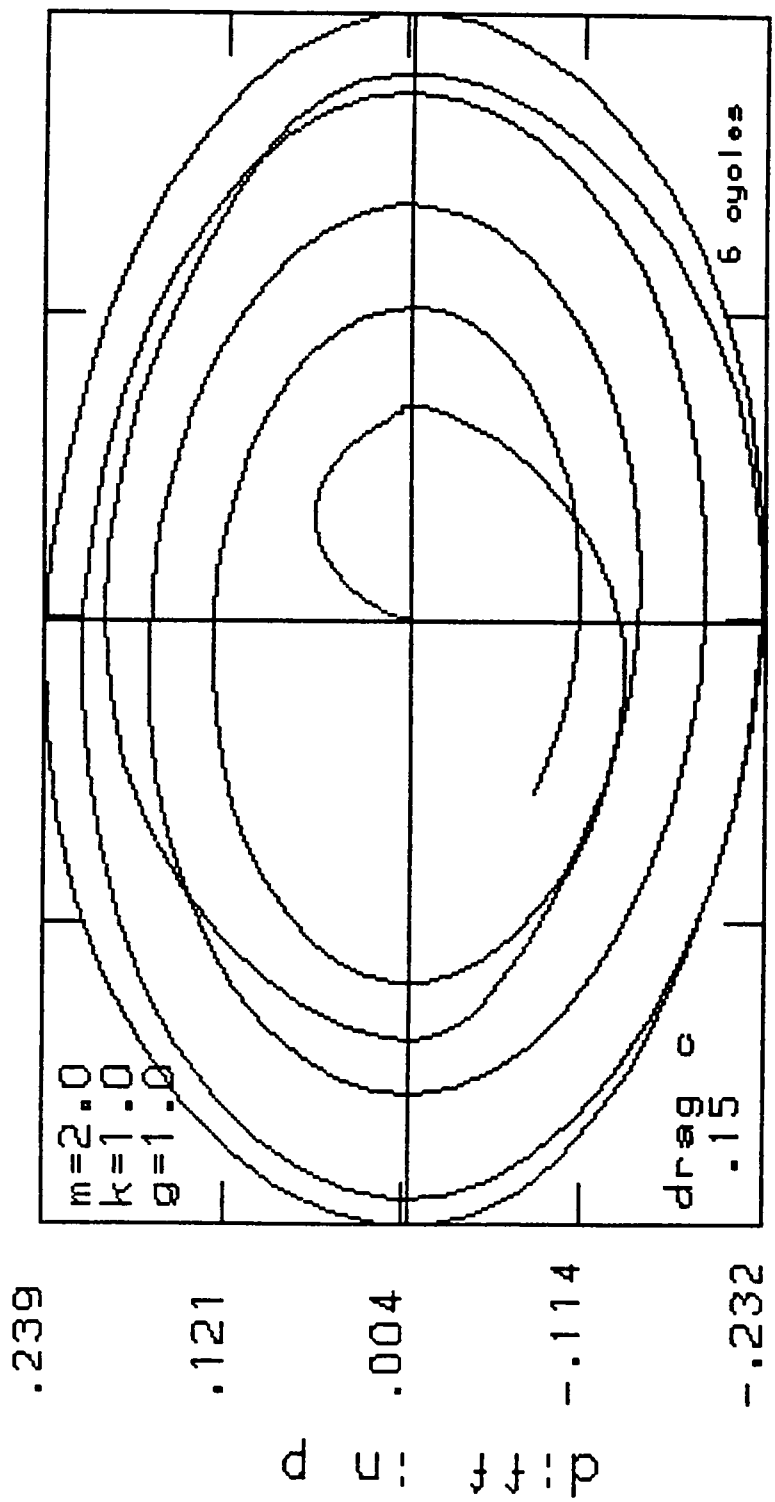


PHASE PLOT exact & 9x9 approx d=.15

Figure 4.1

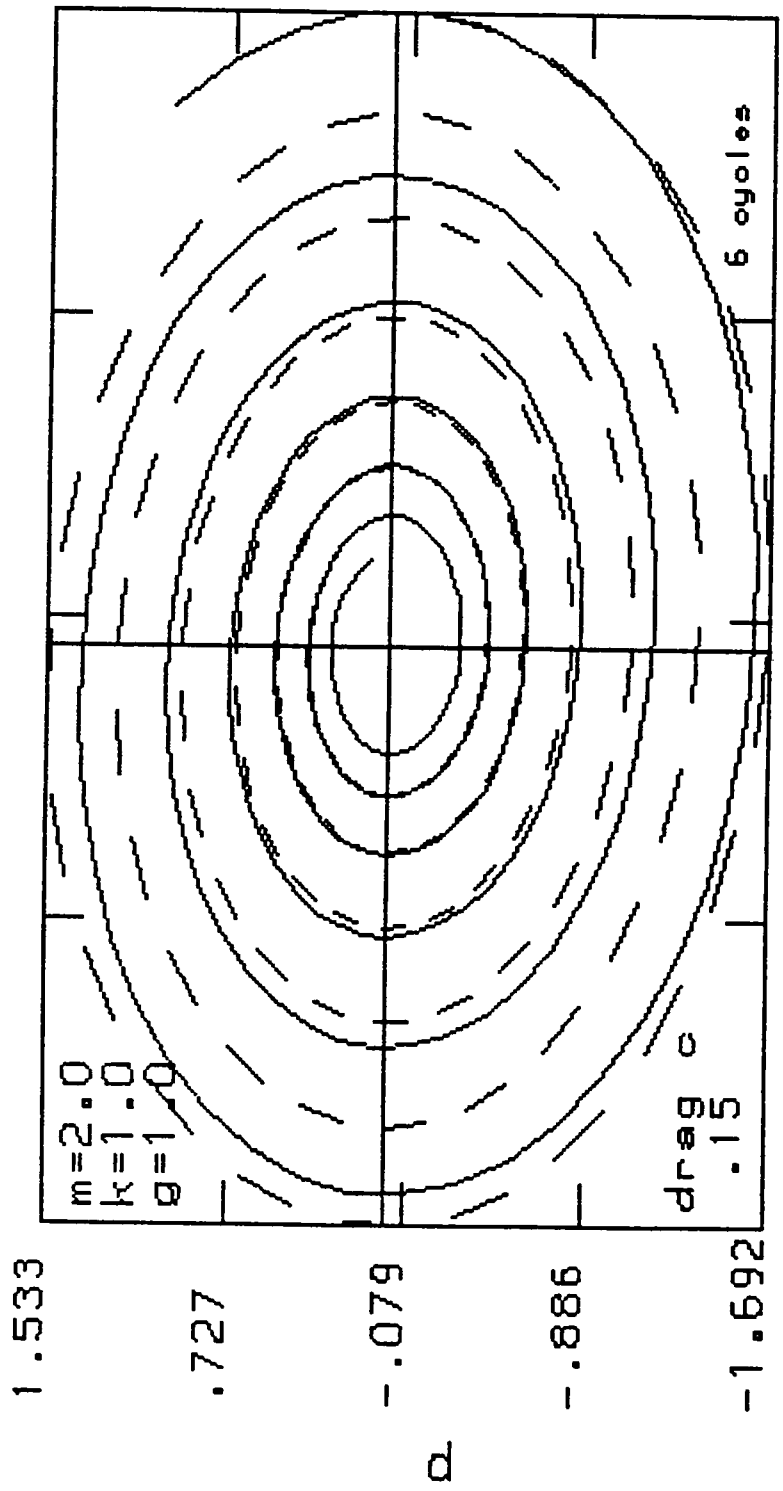
existence of a certain amount of attenuation in the magnitudes of both the position and momentum. A plot in phase space of the differences in x and p between the exact and the approximated solutions is exhibited in Figure 4.2 for the same 6 cycles of data. Associated statistics for this 9×9 truncation are presented in Table 4.2 .

For the 14×14 truncation a comparative plot in phase space of x and p from the exact RKG solution (solid line) and the approximation using the Putzer algorithm (dashed line) are presented in Figure 4.3 for 6 cycles of data using a drag coefficient of .15 . The difference from the 9×9 truncation is small. The motion is spirally inward with about the same degree of attenuation of the magnitudes as in the 9×9 truncation case. Differences in phase space in x and p for this case are presented in Figure 4.4 for 6 cycles of data. The associated statistics for Figure 4.3 are contained in Table 4.3, presented below.



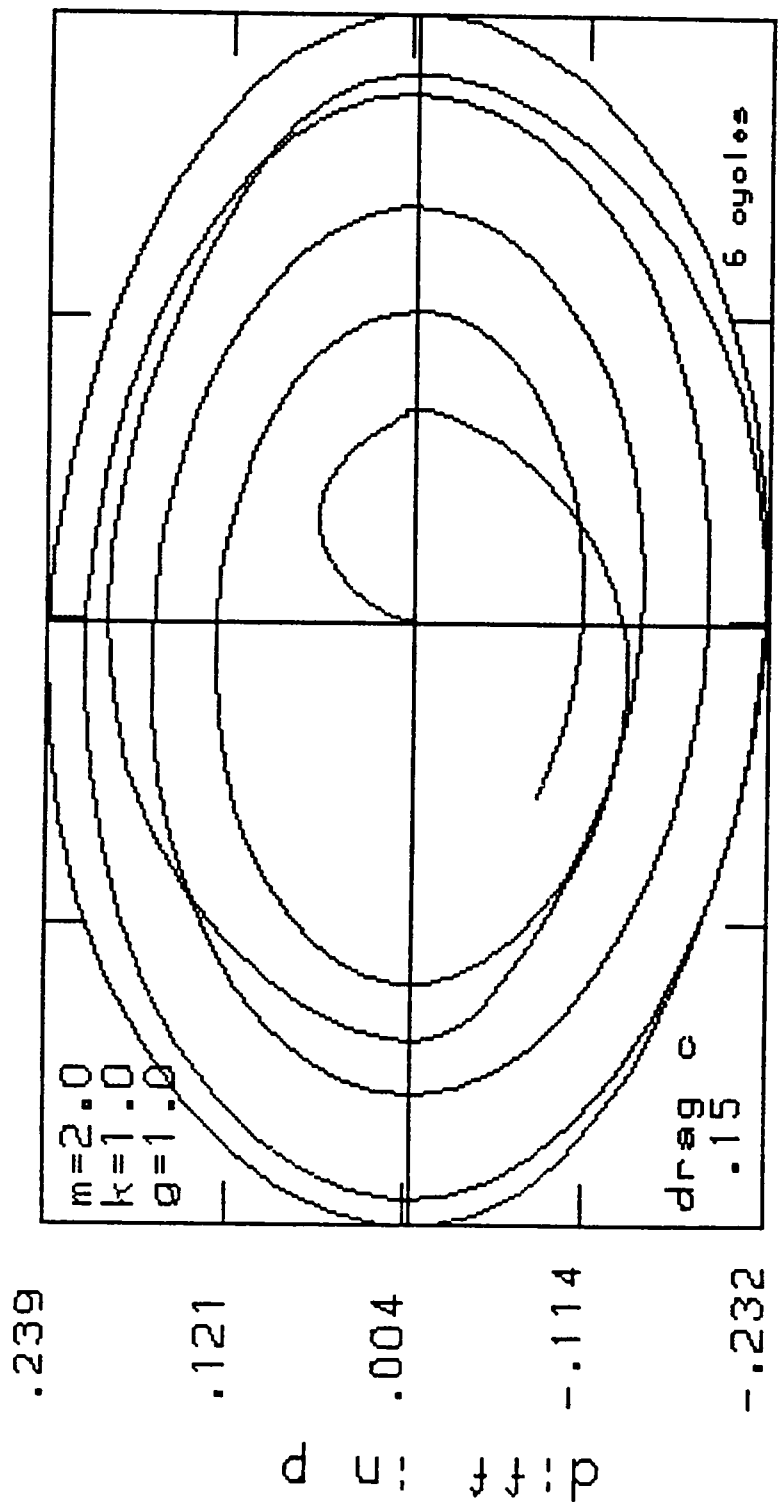
PHASE DIFF exact & 9x9 approx d=.15

Figure 4.2



PHASE PLOT exact & 14x14 approx d=.15

Figure 4.3



PHASE DIFF exact & 14x14 approx d=.15

Figure 4.4

Table 4.2

Statistics for Δx and Δp for exact solution and 9×9 approximation using Putzer algorithm with RKG integration for Drag Coefficient of .15

$m = 2.0 \qquad k = 1.0 \qquad g = 1.0$

$$\alpha = .15 \qquad \beta = \sqrt{\left[\frac{\alpha}{m}\right]^2 - 4\,\frac{k}{m}} = .706111$$

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.043757	.065243
2	86	.076733	.114681
3	130	.092729	.134221
4	174	.096441	.137308
5	218	.094355	.133227
6	263	.089970	.126527

Table 4.3

Statistics for Δx and Δp for exact solution and 14×14 approximation using Putzer algorithm with RKG integration for Drag Coefficient of .15

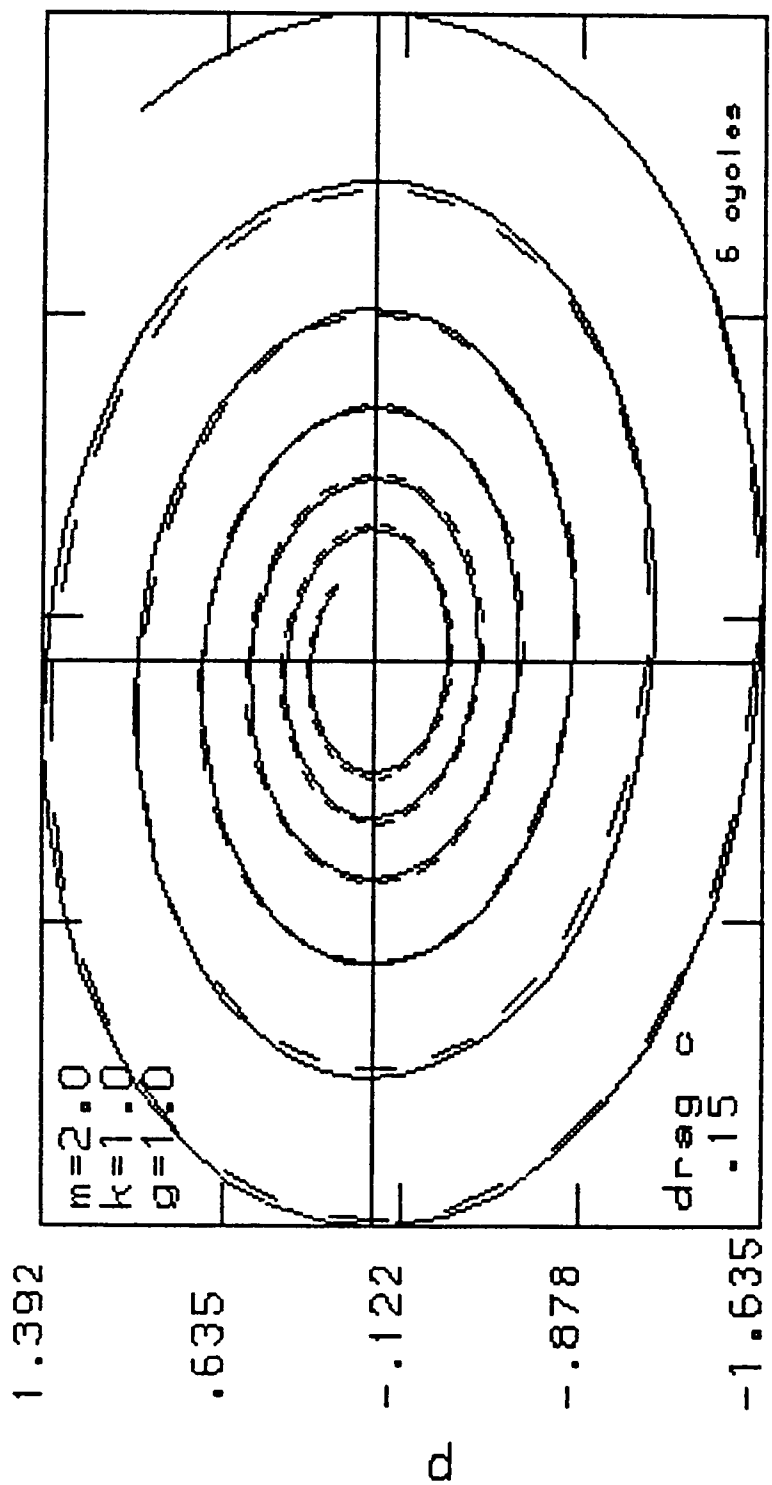
$m = 2.0 \qquad k = 1.0 \qquad g = 1.0$

$$\alpha = .15 \qquad \beta = \sqrt{\left[\frac{\alpha}{m}\right]^2 - 4\,\frac{k}{m}} = .706111$$

number of cycles	Number of data points	1 σ values	
		position	momentum
1	42	.043757	.065243
2	86	.076733	.114681
3	130	.092729	.134222
4	174	.096441	.137308
5	218	.094355	.133227
6	263	.089970	.126527

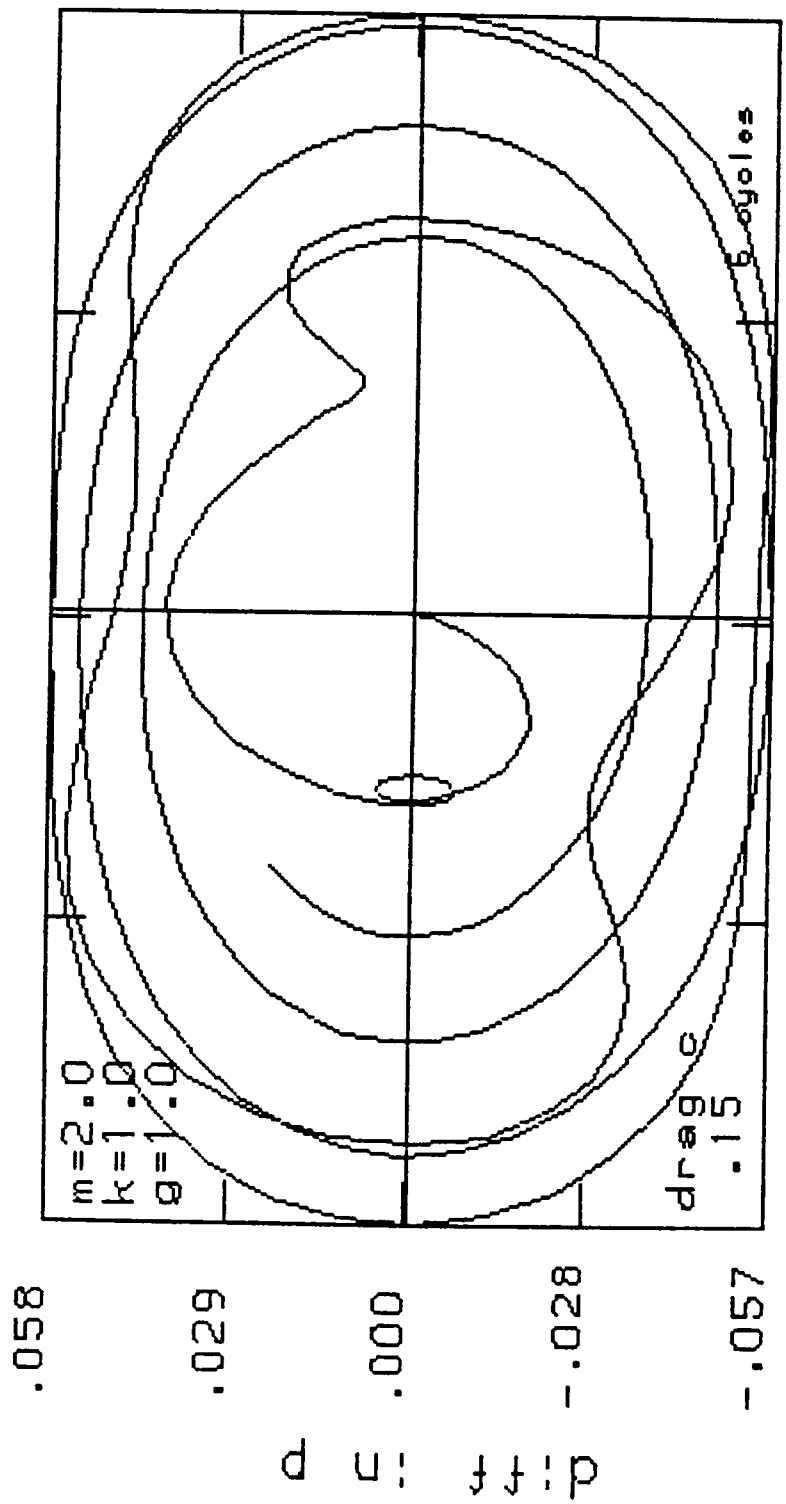
The data produced by the 20×20 truncation of the L matrix are examined next. A comparative plot in phase space for x and p from the exact RKG solution (solid line) and the approximation (dashed line) is presented in Figure 4.5 for 6 cycles of data. A drag coefficient of 0.15 was used. The attenuation observed in the earlier truncations is still present. The phase space differences of x and p for the same 6 cycles of data of Figure 4.5 for this 20×20 truncation are displayed in Figure 4.6 . The differences are centralized and bounded over the 6 cycles of data. Table 4.4 contains the statistics associated with Figure 4.5 . Appendix D contains the FORTRAN source listing for the eigenvalue calculation for the 54x54 truncation of the L matrix of the perturbed simple harmonic oscillator problem under the influence of a drag force. Appendix E presents the FORTRAN source listing for the approximation scheme employing the Putzer algorithm for the 20×20 L matrix truncation for the above problem. Appendix F contains the FORTRAN source listing for the Runge-Kutta Gill numerical integrator used in obtaining the exact results for the same problem.

A summary of how the standard deviations of the errors in x and p change when progressively larger truncations of the L matrix are made while implementing the approximation exploiting the Putzer algorithm is graphically presented in



PHASE PLOT exact & 20x20 approx d=.15

Figure 4.5



PHASE DIFF exact & 20x20 approx d=.15

Figure 4.6

Table 4.4

Statistics for Δx and Δp for exact solution and 20×20 approximation using Putzer algorithm with RKG integration for Drag Coefficient of .15

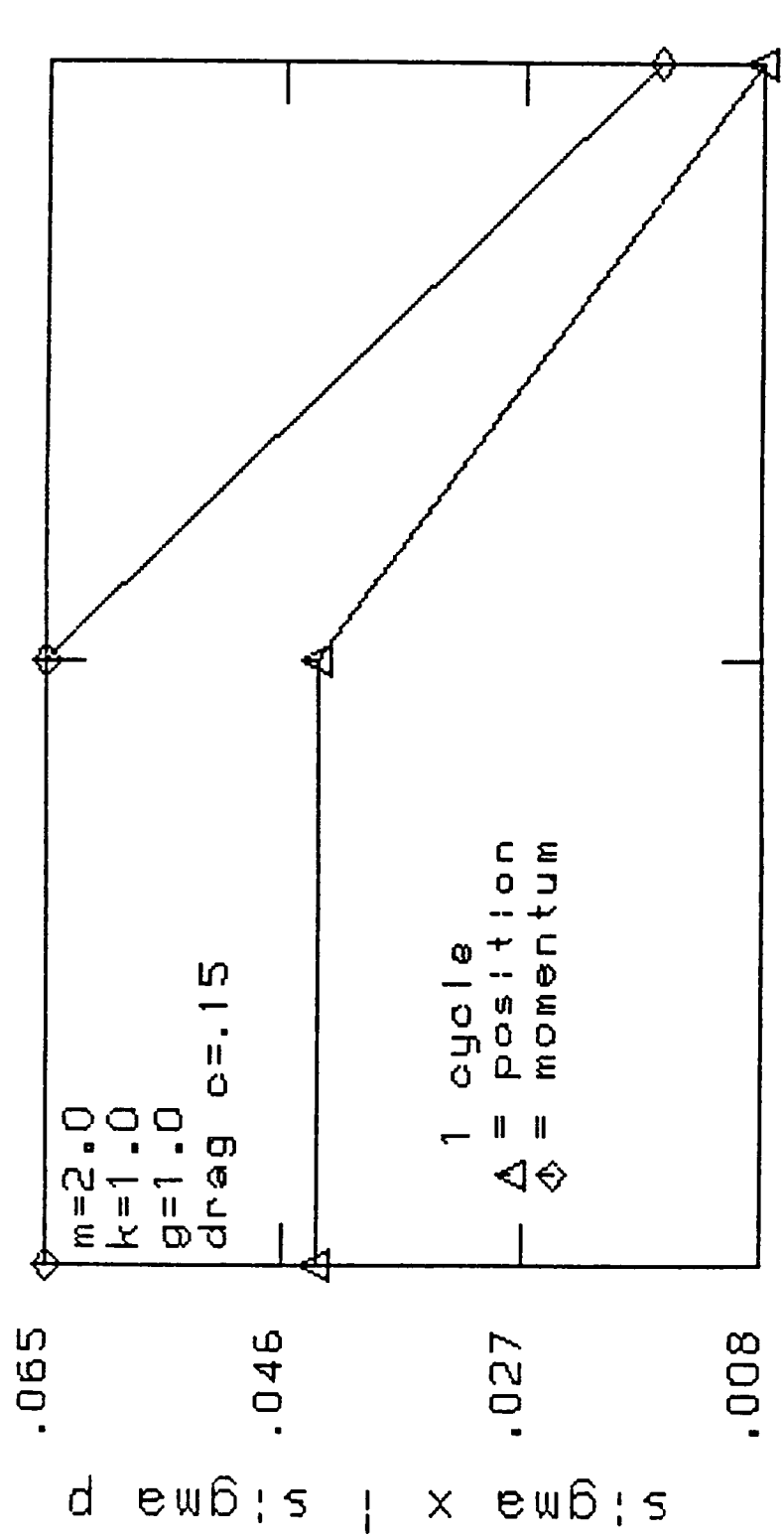
$m = 2.0 \qquad k = 1.0 \qquad g = 1.0$

$$\alpha = .15 \qquad \beta = \sqrt{\left[\frac{\alpha}{m}\right]^2 - 4\,\frac{k}{m}} = .706111$$

number of cycles	Number of data points	1 σ values position	momentum
1	43	.008494	.016451
2	85	.019094	.026005
3	128	.024773	.032638
4	172	.026560	.035846
5	216	.026403	.036185
6	261	.025401	.035085
7	300	.024257	.033755

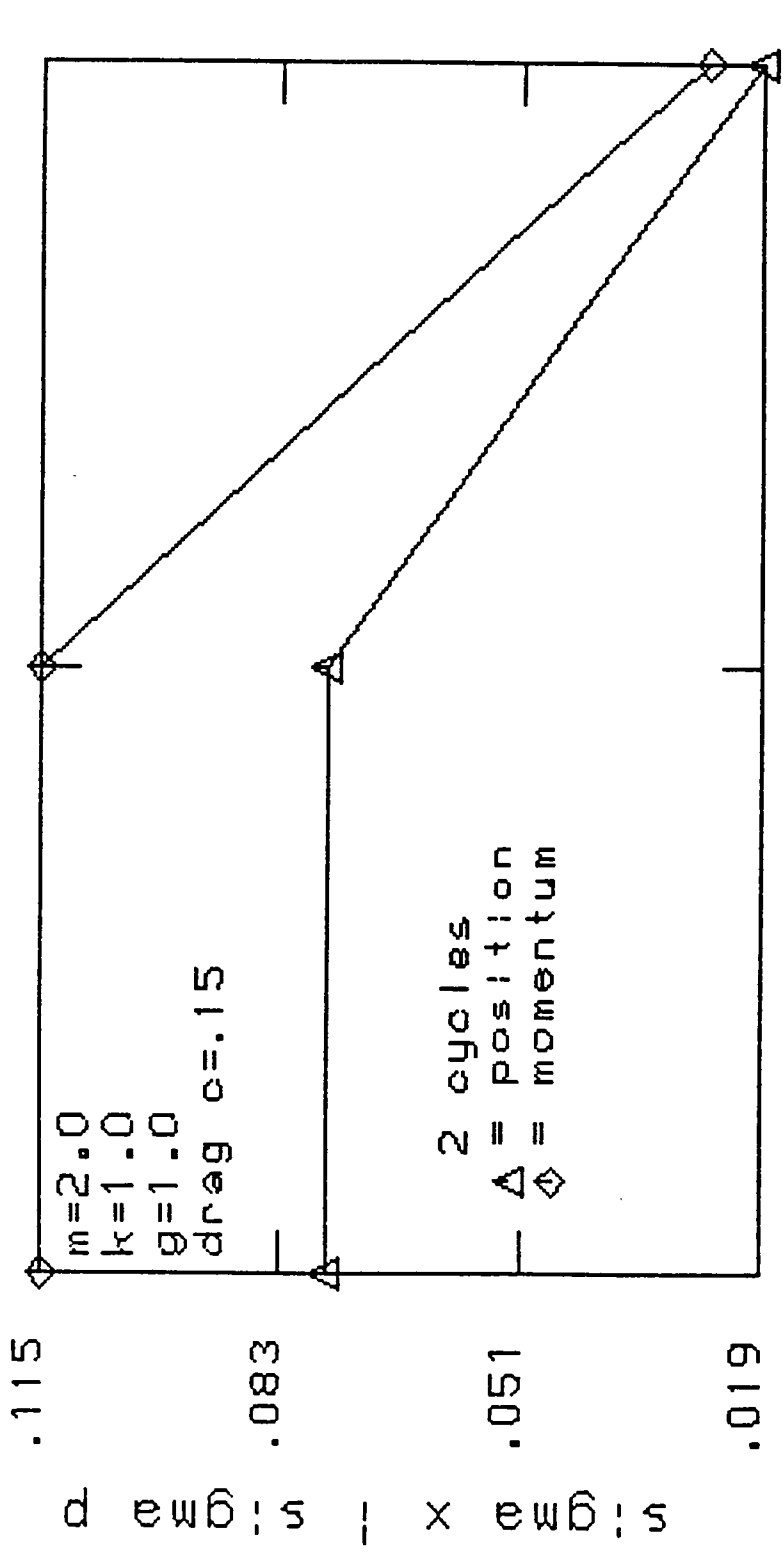
the series of Figures 4.7 through 4.12, covering data for 1 cycle, 2 cycles, 3 cycles, 4 cycles, 5 cycles, and 6 cycles respectively. Truncation sizes range from the 9×9 to the 20×20 in each figure.

The most notable behavior of the statistical errors in this set of figures is that, regardless of how many cycles are considered, there is virtually no change in the statistics for the 14×14 truncation of the L matrix when compared to the statistics for the 9×9 truncation. One would naturally expect a decline in the magnitude of the standard deviations of the errors when the larger 14×14 truncation is considered. Table 4.5 presents the numerical values of the eigenvalues used in generating the data contained in the above figures. The structure is the same as that shown in Table 4.1 . Examination of the eigenvalues reveals that not all of the previously encountered eigenvalues are repeated in each partition. For example, the second partition consisting of eigenvalues numbered 3, 4, 5 does not contain eigenvalues numbered 1 and 2 that were found in the *first* partition. Similarly, the third partition consisting of eigenvalues numbered 6-9 repeats the eigenvalues 1 and 2 of the first partition and adds a new conjugate pair of complex numbers to the eigenvalue list while not repeating the eigenvalues labeled 3, 4, 5 of the *second* partition. The fourth partition



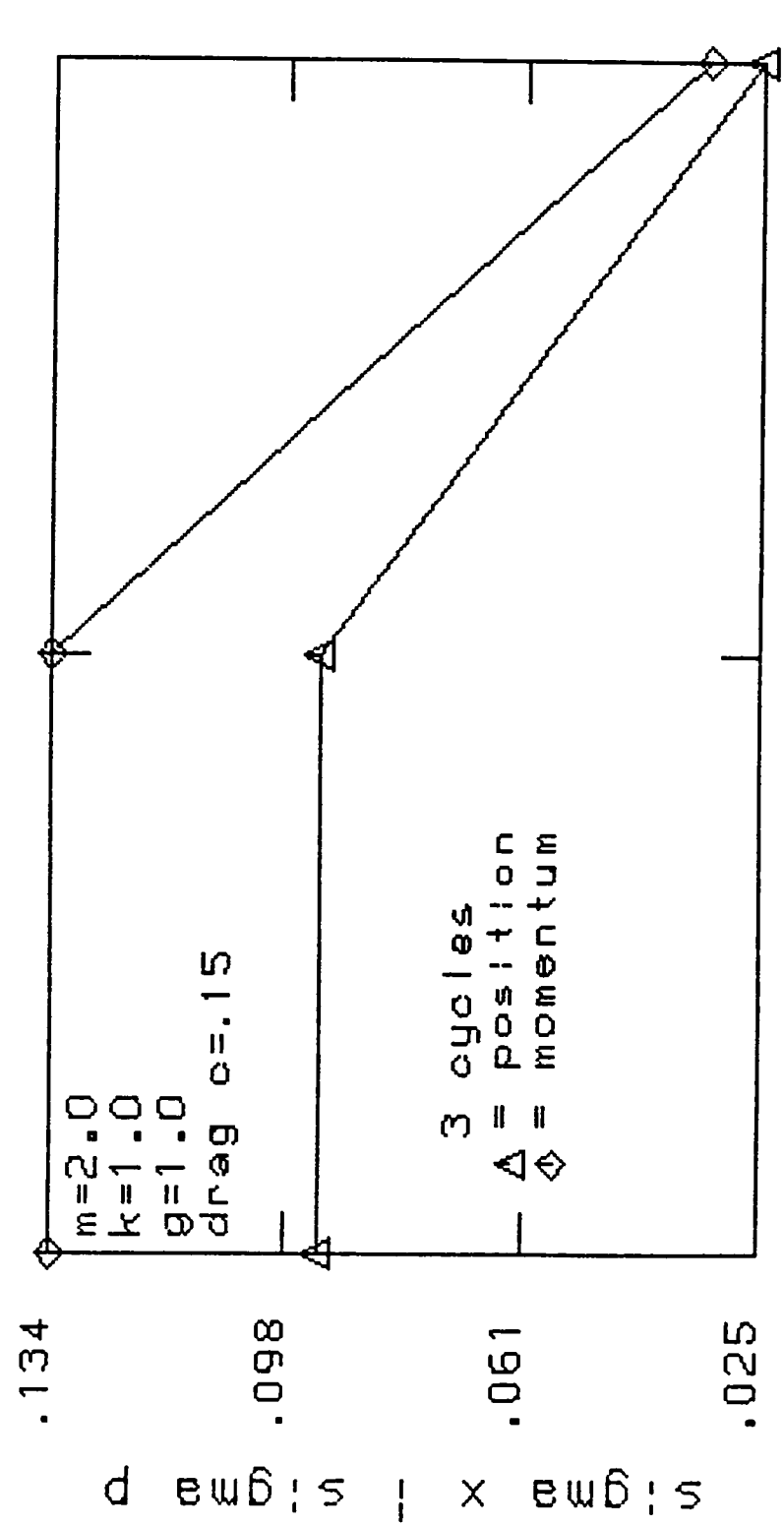
STATISTICS by Truncation Size drag.15

Figure 4.7



STATISTICS by Truncation Size drag.15

Figure 4.8



STATISTICS by Truncation Size drag.15

Figure 4.9

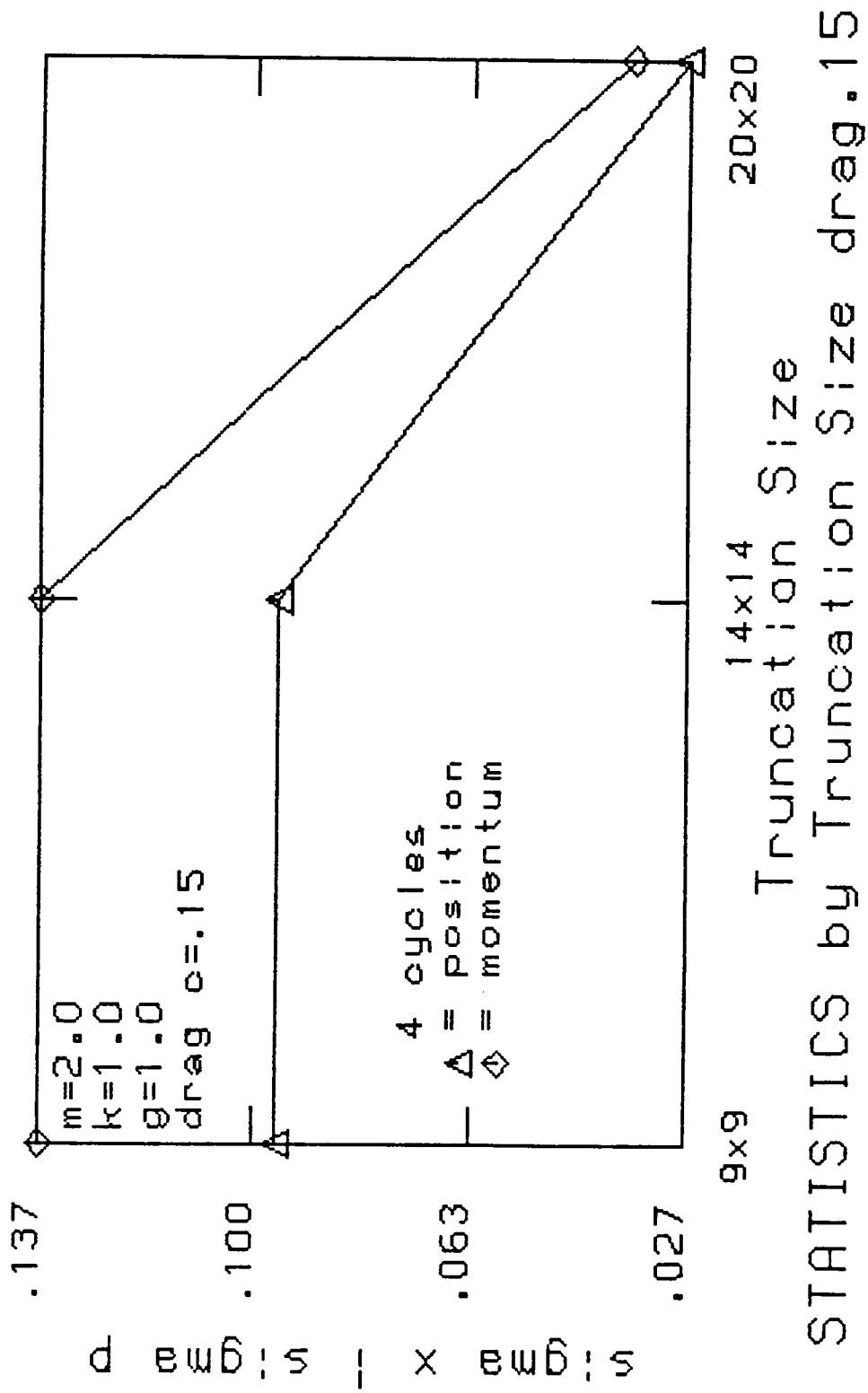
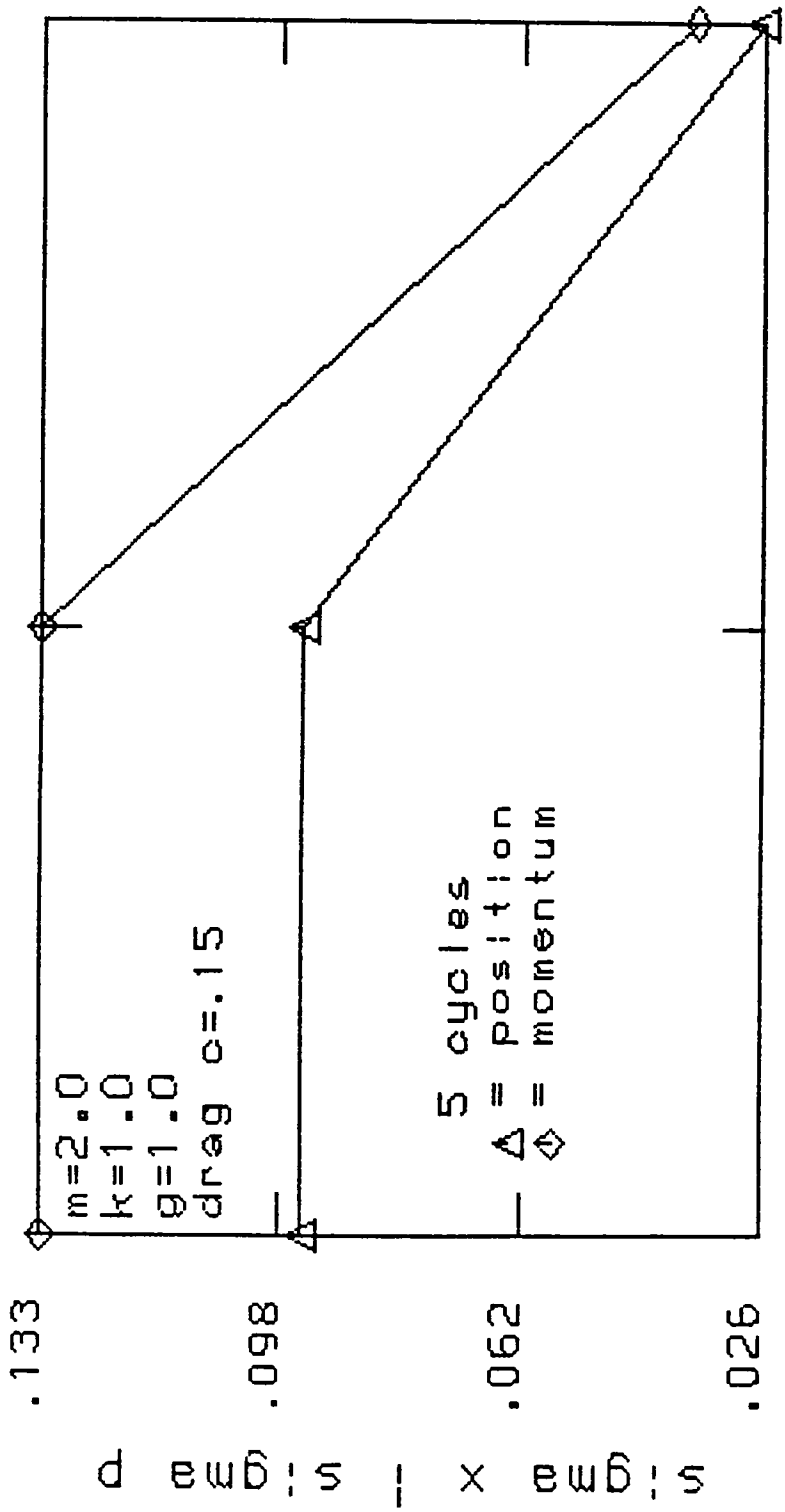

STATISTICS by Truncation Size drag.15
Truncation Size
9x9
14x14
20x20
.137
.100
.063
.027
sigma x | sigma p
m=2.0
k=1.0
g=1.0
drag c=.15
4 cycles
Δ = position
◇ = momentum

Figure 4.10



STATISTICS by Truncation Size drag.15

Figure 4.11

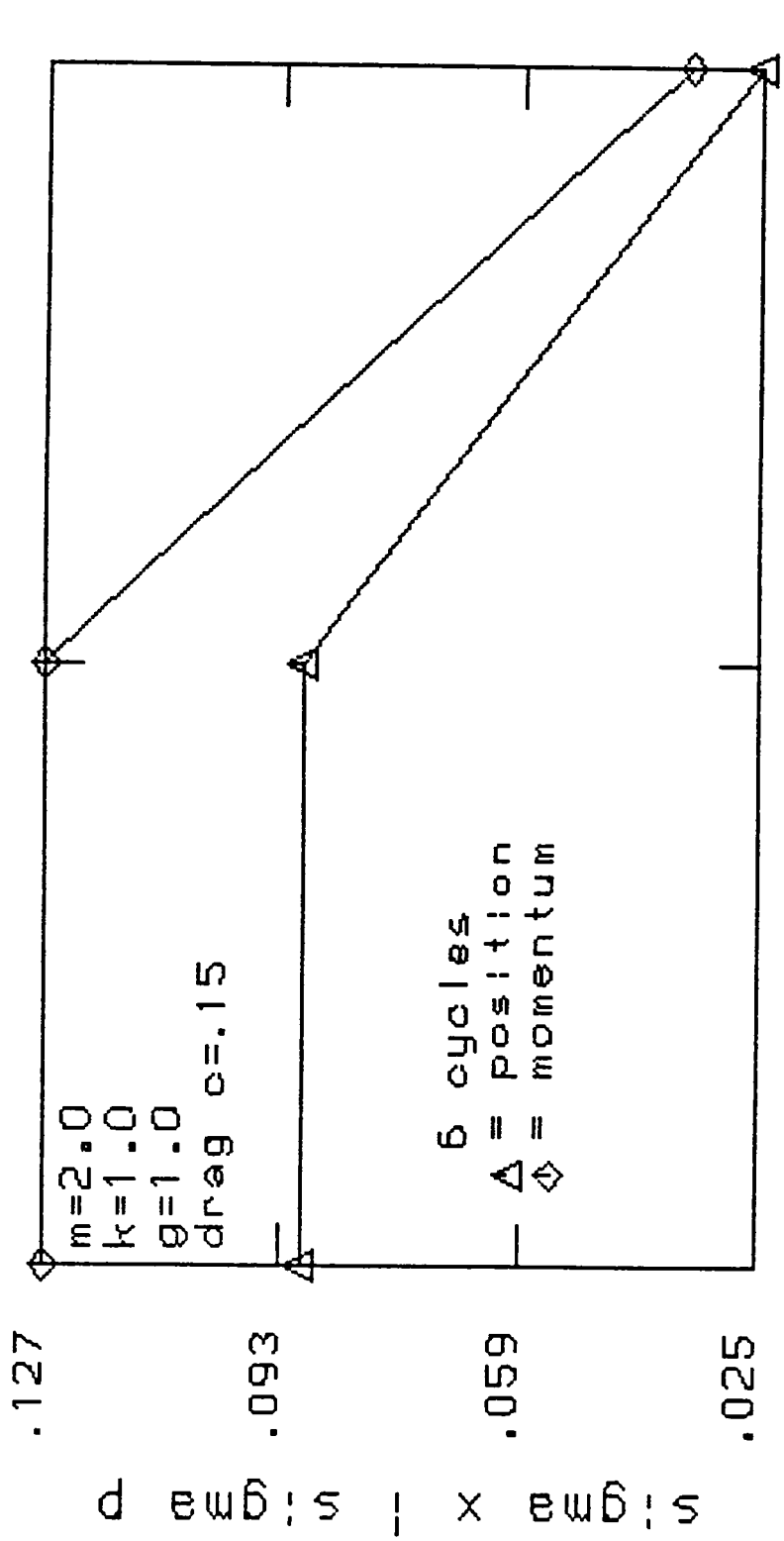

STATISTICS by Truncation Size drag.15
Truncation Size
9x9
14x14
20x20
sigma x | sigma p
.127
.093
.059
.025
m=2.0
k=1.0
g=1.0
drag c=.15
6 cycles
= position
= momentum

Figure 4.12

Table 4.5

Numerical Eigenvalues of L matrix for up to 54×54 truncation; m=2, k=1, g=1

$$\alpha = .15 \qquad \beta = \sqrt{\left[\frac{\alpha}{m}\right]^2 - 4\,\frac{k}{m}} = .706111$$

ℓ^{th} Eigenvalue	Real Part	Imaginary Part
i= 1 EIGENVALUE:	-.37500000000D-01	.7061117121D+00(I)
i= 2 EIGENVALUE:	-.37500000000D-01	-.7061117121D+00(I)
i= 3 EIGENVALUE:	-.75000000000D-01	.1412223424D+01(I)
i= 4 EIGENVALUE:	-.75000000000D-01	-.1412223424D+01(I)
i= 5 EIGENVALUE:	-.75000000000D-01	.00000000000D+00(I)
i= 6 EIGENVALUE:	-.11250000000D+00	.2118335136D+01(I)
i= 7 EIGENVALUE:	-.11250000000D+00	-.2118335136D+01(I)
i= 8 EIGENVALUE:	-.11250000000D+00	.7061117121D+00(I)
i= 9 EIGENVALUE:	-.11250000000D+00	-.7061117121D+00(I)
i= 10 EIGENVALUE:	-.15000000000D+00	.2824446848D+01(I)
i= 11 EIGENVALUE:	-.15000000000D+00	-.2824446848D+01(I)
i= 12 EIGENVALUE:	-.15000000000D+00	.1412223424D+01(I)
i= 13 EIGENVALUE:	-.15000000000D+00	-.1412223424D+01(I)
i= 14 EIGENVALUE:	-.15000000000D+00	.00000000000D+00(I)
i= 15 EIGENVALUE:	-.18750000000D+00	.3530558561D+01(I)
i= 16 EIGENVALUE:	-.18750000000D+00	-.3530558561D+01(I)
i= 17 EIGENVALUE:	-.18750000000D+00	.2118335136D+01(I)
i= 18 EIGENVALUE:	-.18750000000D+00	-.2118335136D+01(I)
i= 19 EIGENVALUE:	-.18750000000D+00	.7061117121D+00(I)
i= 20 EIGENVALUE:	-.18750000000D+00	-.7061117121D+00(I)
i= 21 EIGENVALUE:	-.22500000000D+00	.4236670273D+01(I)
i= 22 EIGENVALUE:	-.22500000000D+00	-.4236670273D+01(I)
i= 23 EIGENVALUE:	-.22500000000D+00	.2824446848D+01(I)
i= 24 EIGENVALUE:	-.22500000000D+00	-.2824446848D+01(I)
i= 25 EIGENVALUE:	-.22500000000D+00	.1412223424D+01(I)
i= 26 EIGENVALUE:	-.22500000000D+00	-.1412223424D+01(I)
i= 27 EIGENVALUE:	-.22500000000D+00	.00000000000D+00(I)
i= 28 EIGENVALUE:	-.26250000000D+00	.4942781985D+01(I)
i= 29 EIGENVALUE:	-.26250000000D+00	-.4942781985D+01(I)
i= 30 EIGENVALUE:	-.26250000000D+00	.3530558561D+01(I)
i= 31 EIGENVALUE:	-.26250000000D+00	-.3530558561D+01(I)
i= 32 EIGENVALUE:	-.26250000000D+00	.2118335136D+01(I)
i= 33 EIGENVALUE:	-.26250000000D+00	-.2118335136D+01(I)
i= 34 EIGENVALUE:	-.26250000000D+00	.7061117121D+00(I)
i= 35 EIGENVALUE:	-.26250000000D+00	-.7061117121D+00(I)

Table 4.5 (Cont)

Numerical Eigenvalues of L matrix for up to 54×54 truncation; m=2, k=1, g=1

$$\alpha = .15 \qquad \beta = \sqrt{\left[\frac{\alpha}{m}\right]^2 - 4\,\frac{k}{m}} = .706111$$

i^{th} Eigenvalue	Real Part	Imaginary Part
i= 36 EIGENVALUE:	-.3000000000D+00	.5648893697D+01(I)
i= 37 EIGENVALUE:	-.3000000000D+00	-.5648893697D+01(I)
i= 38 EIGENVALUE:	-.3000000000D+00	.4236670273D+01(I)
i= 39 EIGENVALUE:	-.3000000000D+00	-.4236670273D+01(I)
i= 40 EIGENVALUE:	-.3000000000D+00	.2824446848D+01(I)
i= 41 EIGENVALUE:	-.3000000000D+00	-.2824446848D+01(I)
i= 42 EIGENVALUE:	-.3000000000D+00	.1412223424D+01(I)
i= 43 EIGENVALUE:	-.3000000000D+00	-.1412223424D+01(I)
i= 44 EIGENVALUE:	-.3000000000D+00	.0000000000D+00(I)
i= 45 EIGENVALUE:	-.3375000000D+00	.6355005409D+01(I)
i= 46 EIGENVALUE:	-.3375000000D+00	-.6355005409D+01(I)
i= 47 EIGENVALUE:	-.3375000000D+00	.4942781985D+01(I)
i= 48 EIGENVALUE:	-.3375000000D+00	-.4942781985D+01(I)
i= 49 EIGENVALUE:	-.3375000000D+00	.3530558561D+01(I)
i= 50 EIGENVALUE:	-.3375000000D+00	-.3530558561D+01(I)
i= 51 EIGENVALUE:	-.3375000000D+00	.2118335136D+01(I)
i= 52 EIGENVALUE:	-.3375000000D+00	-.2118335136D+01(I)
i= 53 EIGENVALUE:	-.3375000000D+00	.7061117121D+00(I)
i= 54 EIGENVALUE:	-.3375000000D+00	-.7061117121D+00(I)

consisting of eigenvalues numbered 10 through 14 repeats the eigenvalues of the second partition and adds a new conjugate pair of numbers to the eigenvalue list omitting the eigenvalues labeled 6-9 of the *third* partition. This eigenvalue placement among the partitions continues for higher truncations of the L matrix. **This behavior of the eigenvalues was not present for the non-drag case. For the non-drag case each new partition of the L matrix repeated all previous eigenvalues before a new eigenvalue pair was introduced.** (See Chapter 3). This placement of the eigenvalues determines the constant nature of the standard deviations between the 9×9 truncation and the 14×14 truncation of the L matrix containing drag terms. The standard deviations do decrease monotonically for both the position and the momentum in the expected way when the 20×20 truncation is compared with the 14×14 truncation of the L matrix for the drag case. Although only three truncations were used to generate data, the anticipation is that this "staircase" error behavior will be continued for truncations greater than the 20×20.

As can be seen from the phase space plots, the presence of drag acts to improve the quality of the truncation approximations in the sense that the polynomial time behavior characteristic of a finite truncation is suppressed by the damping of the drag part of the force.

The significance of the results of this chapter is that the successive approximation of effective L matrix truncations does lead to better approximations as the truncation dimension is increased. The presence of non-conservative forces gives this truncation scheme a wider generality than would have been otherwise obtained.

In the next chapter the original perturbed Hamiltonian of Chapter 3 will be modified to include a simple non-polynomial function describing the potential encountered by a physical pendulum. The objective will be to examine the behavior of the approximation scheme and the Liouville operator when simple non-polynomial functions are used to describe the potential. The initially defined Liouville operator will be used. Different sets of bases vectors will be considered and matrices will be produced and analyzed.

CHAPTER 5

EXPLORATION OF APPROXIMATION TECHNIQUE WITH PUTZER ALGORITHM ON HAMILTONIANS HAVING SIMPLE FUNCTIONAL DEPENDENCE

Heretofore this investigation has utilized potentials that were written in terms of monomial and polynomial expressions. During application of the approximation scheme to these potentials the Liouville operator produced previously encountered term(s) and generated new terms. The task of recognizing these terms is easiest when monomial and polynomial forms are chosen to express the potentials. The question remains as to whether the approximation technique utilizing the Liouville operator will work when a simple non-polynomial function is used in lieu of monomial and polynomial functions.

The preceding truncation approximations have all used polynomials in x and p. This chapter will investigate different sets of functions of x and p as approximation basis sets. A simple non-polynomial function of x, cos(x), will be used. The Hamiltonian is written in the standard form for a physical pendulum

$$H = \frac{p^2}{2m} + mf\left[1 - \cos(x)\right] \quad ; \; f \equiv Gl, \qquad (5.1)$$

where x is a small angular displacement of a physical pendulum, and l, is the length from the center of mass to

the pivotal point. Immediately the problem of what to choose for the initial basis of independent vectors is encountered. For a first attempt, x and p are initially chosen for the basis vectors, so that the Liouville operator $\mathcal{L}$ then generates the following set of vectors:

$$\mathcal{L}(x) = \frac{1}{m} p \tag{5.2}$$

$$\mathcal{L}(p) = -mf \sin(x) \tag{5.3}$$

$$\mathcal{L}(\sin(x)) = \frac{1}{m} p \cos(x) \tag{5.4}$$

$$\mathcal{L}(p \cos(x)) = -\frac{1}{m} p^2 \sin(x) - \frac{mf}{2} \sin(2x) \tag{5.5}$$

$$\mathcal{L}(p^2 \sin(x)) = \frac{1}{m} p^3 \cos(x) - 2mf\, p \sin^2(x) \tag{5.6}$$

$$\mathcal{L}(\sin(2x)) = \frac{2}{m} p \cos(2x) \tag{5.7}$$

$$\mathcal{L}(p^3 \cos(x)) = -\frac{1}{m} p^4 \sin(x) - \frac{3mf}{2} p^2 \sin(2x) \tag{5.8}$$

$$\mathcal{L}(p \sin^2(x)) = \frac{1}{m} p^2 \sin(2x) - mf \sin^3(x) \tag{5.9}$$

When a matrix is formed of the coefficents of the "new" vectors generated, there is no "link back" or connection to previously generated vectors, as was the case when polynomial Hamiltonians were studied. The matrix for the above vectors would look like

	x	p	s(x)	pc(x)	s(2x)	$p^2s(x)$	$ps^2(x)$	pc(2x)	$p^3c(x)$
x	**0**	m^{-1}	0	0	0	0	0	0	0
p	0	**0**	-mf	0	0	0	0	0	0
s(x)	0	0	**0**	1/m	0	0	0	0	0
pc(x)	0	0	0	**0**	-.5mf	-1/m	0	0	0
s(2x)	0	0	0	0	**0**	0	0	1/m	0
$p^2s(x)$	0	0	0	0	0	**0**	-2mf	0	1/m
$ps^2(x)$	0	0	0	0	0	0	**0**	0	0
pc(2x)	0	0	0	0	0	0	0	**0**	0
$p^3c(x)$	0	0	0	0	0	0	0	0	**0**

where the abbreviations s(x), c(x) have been used for sin(x), and cos(x) respectively. In the above matrix, the entries of the main diagonal are shown in boldface. This matrix yields, by inspection, only zero eigenvalues. Hence a new basis must be found, as a matrix that yields only zero eigenvalues is not useful for the approximation scheme under study.

The next group of initial "candidate" basis vectors attempted were the vectors

$$1, \; p, \; e^{ix}, \; e^{-ix}.$$

Here the hope is that, by writing the cosine function in an alternate form, an L matrix can be obtained whose eigenvalues are not all zero. The action of the Liouville operator, $\mathcal{L}$, on this set produces

$$\mathcal{L}(1) = 0 \qquad (5.10)$$

$$\mathcal{L}(p) = i\frac{mf}{2}\left[e^{ix} - e^{-ix}\right] \qquad (5.11)$$

$$\mathcal{L}(e^{ix}) = i\frac{1}{m}\,p\,e^{ix} \qquad (5.12)$$

$$\mathcal{L}(e^{-ix}) = -\frac{1}{m}\,p\,e^{-ix} \qquad (5.13)$$

$$\mathcal{L}(pe^{ix}) = i\frac{1}{m}\,p^2 e^{ix} + i\frac{mf}{2}\,e^{i2x} - i\frac{mf}{2}\,(1) \qquad (5.14)$$

$$\mathcal{L}(pe^{-ix}) = -i\frac{1}{m}\,p^2 e^{-ix} - i\frac{mf}{2}\,e^{-i2x} + i\frac{mf}{2}\,(1) \qquad (5.15)$$

$$\mathcal{L}(p^2e^{ix}) = i\frac{1}{m}\,p^3 e^{ix} + i\,mf\,p\,e^{i2x} - i\,mf\,p \qquad (5.16)$$

$$\mathcal{L}(p^2e^{-ix}) = -i\frac{1}{m}\,p^3 e^{-ix} - i\,mf\,p\,e^{-i2x} + i\,mf\,p \qquad (5.17)$$

$$\mathcal{L}(e^{i2x}) = i\frac{2}{m}\,p\,e^{i2x} \qquad (5.18)$$

$$\mathcal{L}(e^{-i2x}) = -i\frac{2}{m}\,p\,e^{-i2x} \qquad (5.19)$$

$$\mathcal{L}(pe^{i2x}) = i\frac{2}{m}\,p^2 e^{i2x} - i\frac{mf}{2}\,e^{ix} + i\frac{mf}{2}\,e^{i3x} \qquad (5.20)$$

$$\mathcal{L}(pe^{-i2x}) = -i\frac{2}{m}p^2e^{-i2x} + i\frac{mf}{2}e^{-ix} - i\frac{mf}{2}e^{-i3x} \qquad (5.21)$$

Continuation is obvious, but it is more useful to make some observations about the "vectors" introduced, and then to generalize. First, note that if n and j are integers such that $0 \le j \le n-1$, all "vectors" having order n can be written down in the general form

$$e^{inx} \quad ; \quad e^{-inx}$$

$$p^j e^{i(n-j)x} \quad ; \quad p^j e^{-i(n-j)x}$$

$$p^n.$$

The L matrix constructed from this set of basis vectors for the 36×36 truncation has the following structural block form

$$L = \left[\begin{array}{c|c|c|c|c|c} 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \mathbf{A} & \mathbf{A} & 0 & 0 & 0 \\ \hline \mathbf{A} & 0 & \mathbf{B} & \mathbf{B} & 0 & 0 \\ \hline 0 & \mathbf{B} & 0 & \mathbf{C} & \mathbf{H} & 0 \\ \hline 0 & 0 & \mathbf{C} & 0 & \mathbf{D} & \mathbf{E} \\ \hline 0 & 0 & 0 & \mathbf{D} & 0 & \mathbf{E} \end{array}\right],$$

where boldface entries are blocks whose dimensions are dependent upon where they are located in the L matrix.

When all of the blocks are substituted into their respective locations in the structural form of the matrix,

the dimensions of the matrix truncations are proportional to n^2 for the basis above. That basis can be listed as

$$1 \mid p\ e^{i\times}\ e^{-i\times} \mid p^2\ pe^{i\times}\ pe^{-i\times}\ e^{i2\times}\ e^{-i2\times} \mid p^3\ p^2e^{i\times}\ p^2e^{-i\times}\ pe^{i2\times}\ pe^{-i2\times}\ e^{i3\times}\ e^{-i3\times} \mid p^4\ p^3e^{i\times}\ p^3e^{-i\times}\ p^2e^{i2\times}\ p^2e^{-i2\times}\ pe^{i3\times}\ pe^{-i3\times}\ e^{i4\times}\ e^{-i4\times} \mid p^5\ p^4e^{i\times}\ p^4e^{-i\times}\ p^3e^{i2\times}\ p^3e^{-i2\times}\ p^2e^{i3\times}\ p^2e^{-i3\times}\ pe^{i4\times}\ pe^{-i4\times}\ e^{i5\times}\ e^{-i5\times} \mid \ldots\ldots,$$

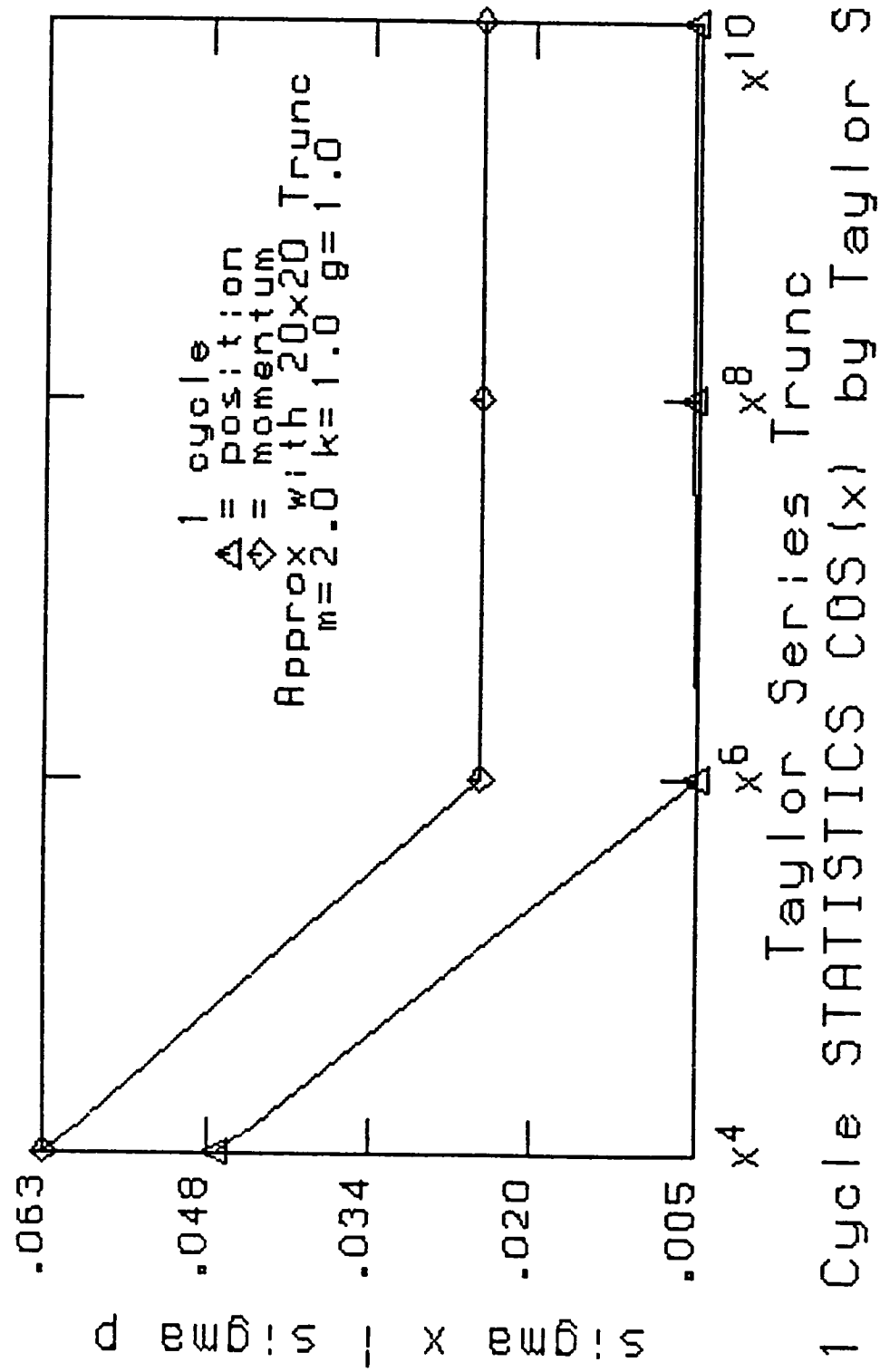
where, for completeness, the basis vectors for up to a 36×36 truncation are shown.

For this basis, the eigenvalue determination of the L matrix for both the 4×4 and the 9×9 truncations of the L matrix using the above basis vectors yield zero eigenvalues. Examination of the next higher truncation (16×16) yielded zero eigenvalues. Proceeding successively to higher truncations of the L matrix, three new blocks are added to the composition of the L matrix for the new row and column partition encountered. Each of the new non-zero blocks have zero rank, and consequently the eigenvalues of the newly constructed truncated matrix are all zero. A zero eigenvalue spectrum gives no information regarding the L matrix as nonzero eigenvalues are needed to obtain the physically realistic approximations for which we are searching. In light of this, further investigative work on the $\cos(x)$ function contained in the Hamiltonian was abandoned. **This represents a major disappointment**

regarding the usefulness of the method, because it reveals that the technique is limited to polynomial approximations.

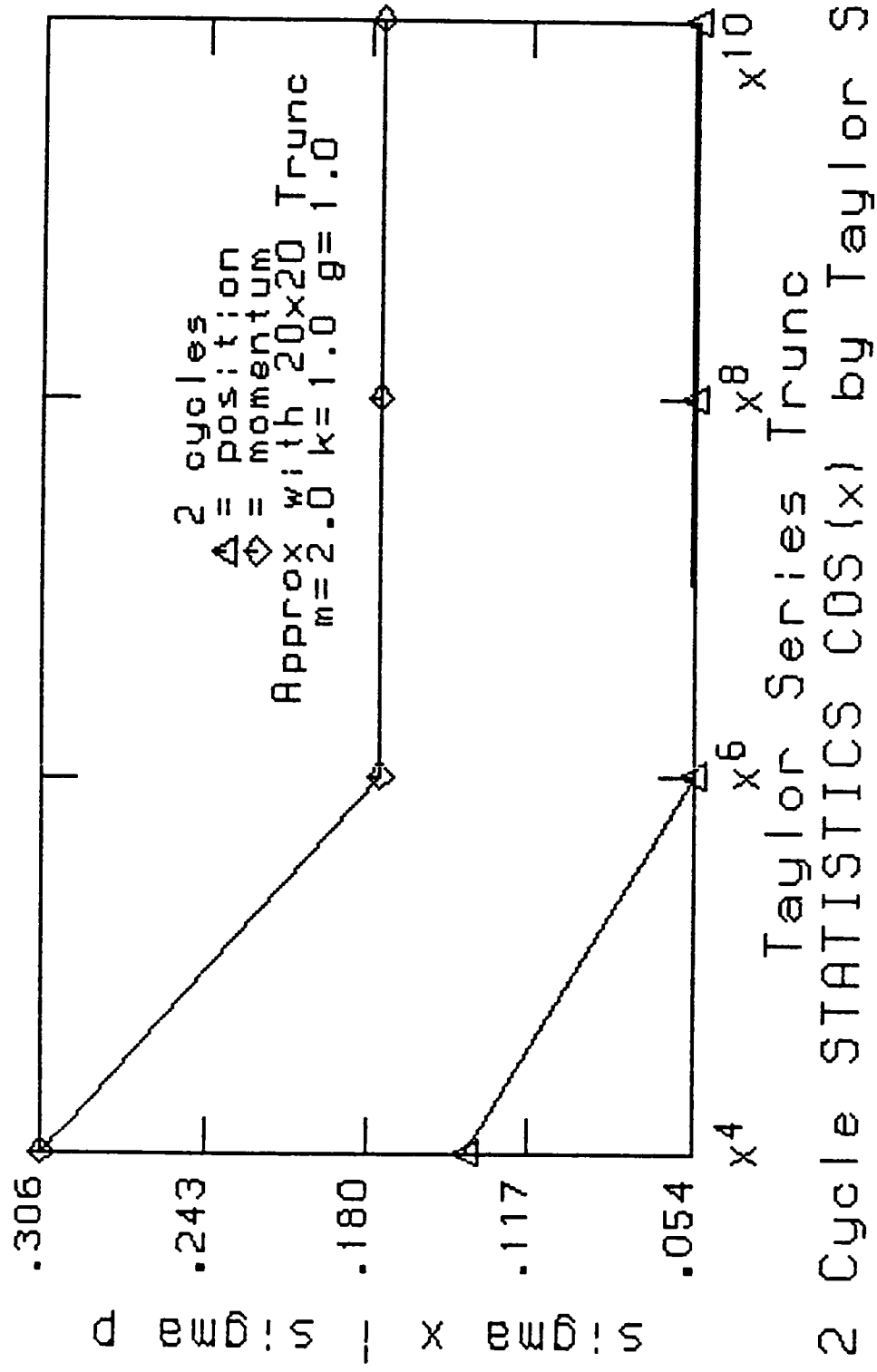
The $\cos(x)$ function in the Hamiltonian was replaced by the corresponding Taylor series in x, which was truncated after the x^{10} term. The Liouville operator $\mathcal{L}$ again produced Hessenberg matrices and all the apparatus previously developed during the study of the prior x^4 Hamiltonian can be applied. For comparison of the Taylor series truncations the $\cos(x)$ function was replaced by, successively, Taylor series having x^{10}, x^8, x^6, and x^4, as the highest degree term of the Taylor series. Now there are two types of truncations embedded in the method. The first type is the truncation of the L matrix itself. The second type of truncation is the termination of the Taylor Series used to approximate the cosine function appearing in the initial Hamiltonian. This truncation determines the placement of non-zero entries in the resultant Hessenberg matrix, and therefore the eigenvalues. Solutions to the exact $\cos(x)$ Hamiltonian were found by Runge-Kutta Gill (RKG) numerical integration. The qualitative nature of the solutions from the approximation method using the Putzer recipe were the same as those discussed in Chapter 3. Consequently a qualitative discussion of the Putzer approximation solutions will not be repeated here.

Figures 5.1 to Figures 5.6 contain comparison plots of



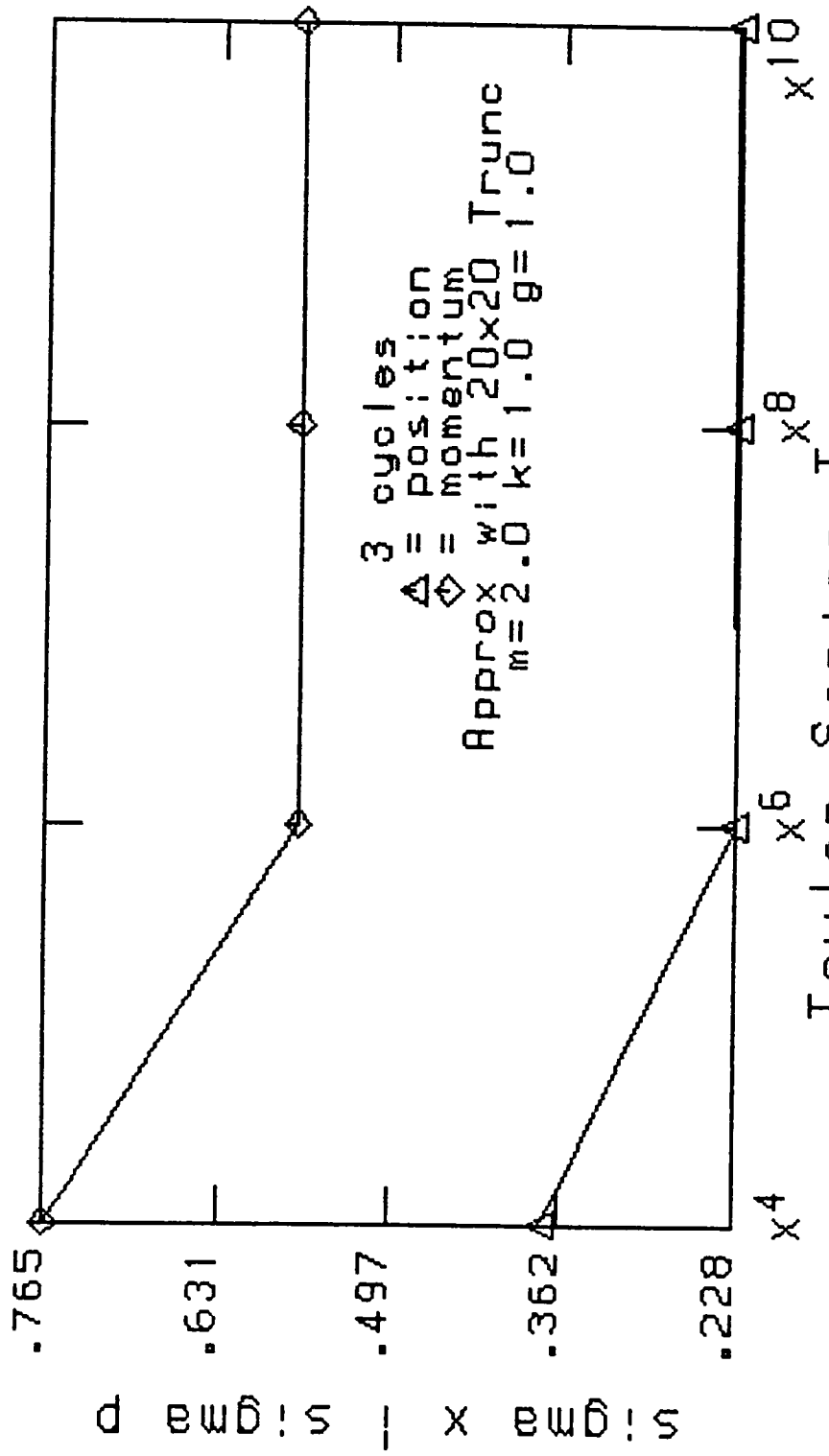
1 Cycle STATISTICS COS(x) by Taylor S

Figure 5.1



2 Cycle STATISTICS COS(x) by Taylor S

Figure 5.2



3 Cycle STATISTICS COS(x) by Taylor Series Trunc

Figure 5.3

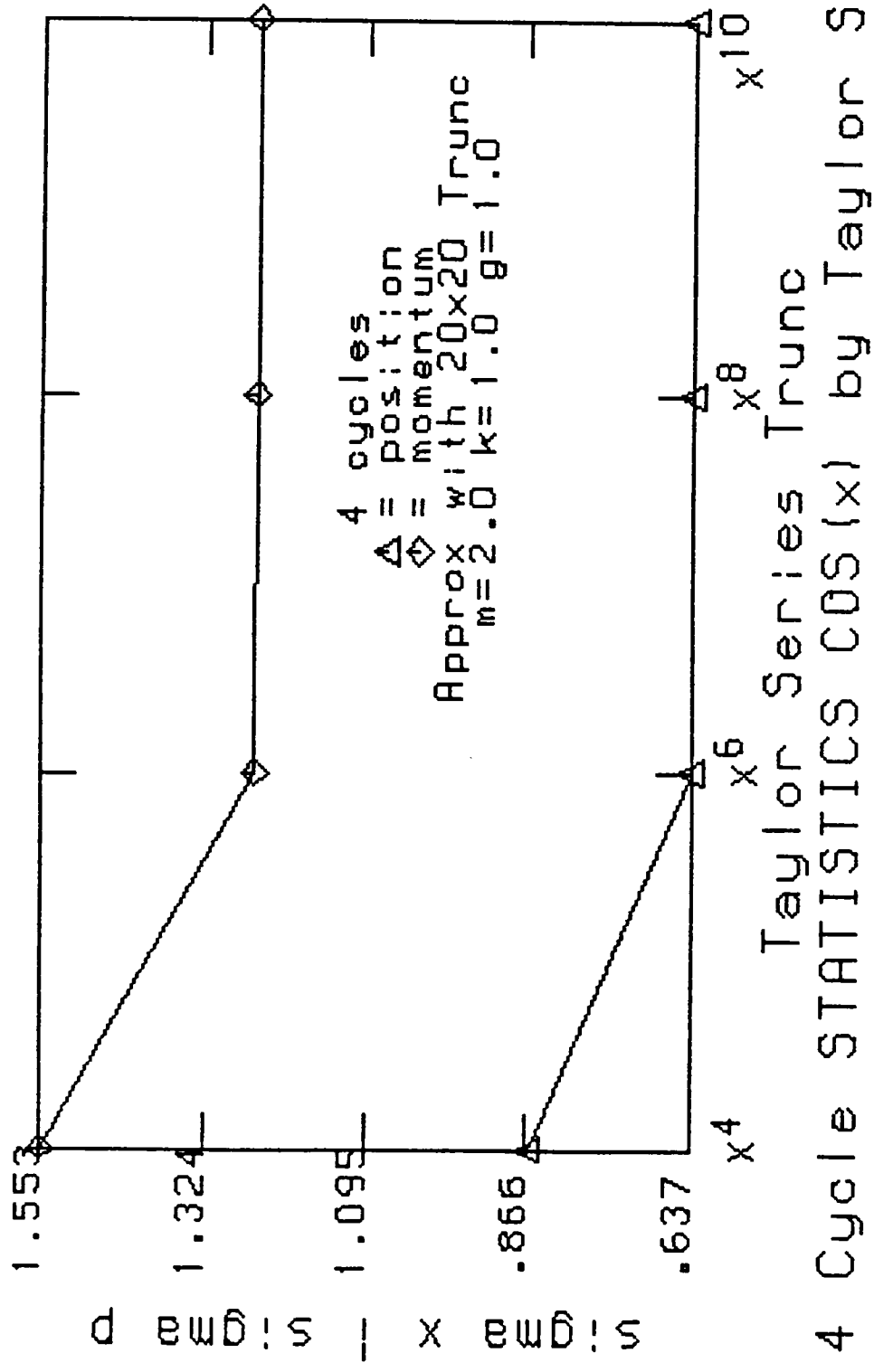

4 Cycle STATISTICS COS(x) by Taylor S
Taylor Series Trunc
4 cycles
= position
= momentum
Approx with 20x20 Trunc
m=2.0 k=1.0 g=1.0
sigma x | sigma p
1.553
1.324
1.095
.866
.637
x4
x6
x8
x10

Figure 5.4

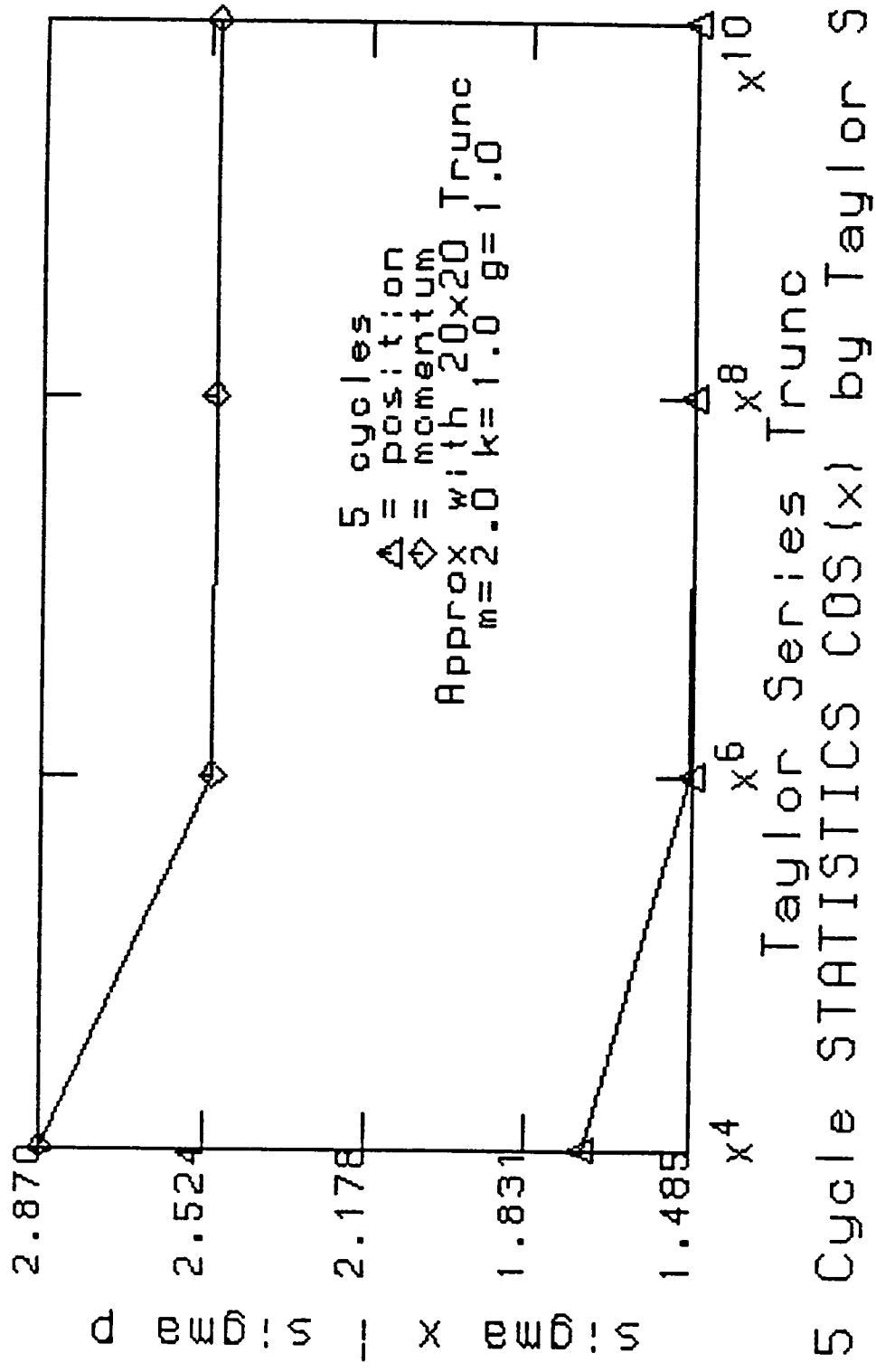
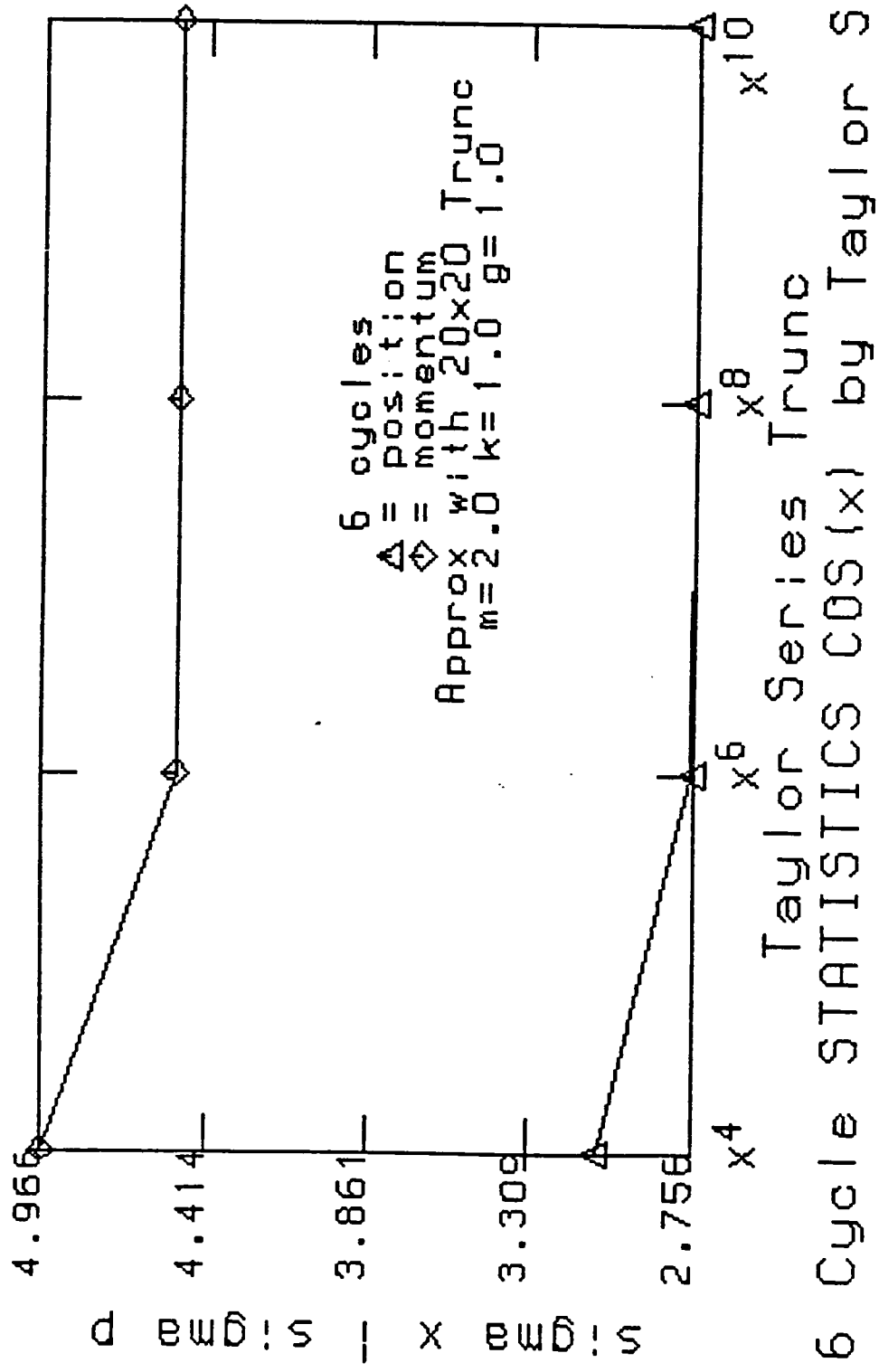

5 Cycle STATISTICS COS(x) by Taylor S
Taylor Series Trunc
sigma x | sigma p
2.870
2.524
2.178
1.831
1.485
x4
x6
x8
x10
5 cycles
= position
= momentum
Approx with 20x20 Trunc
m=2.0 k=1.0 g=1.0

Figure 5.5



6 Cycle STATISTICS COS(x) by Taylor S

Figure 5.6

the statistics of the position, and the momentum, when various orders of Taylor Series truncations are used in the approximation technique. Statistics are compared for each terminating Taylor Series truncation on a cycle by cycle basis. Figure 5.1 through Figure 5.6 present the statistics of Δx and Δp as a function of truncation size for 1 to 6 cycles of data respectively. In all cases there is no change in the statistics produced when the x^6, the x^8, and the x^{10} truncations of the Taylor Series are used. This indicates that there is no benefit in keeping the Taylor series truncations for the cosine function past the sixth order. One explanation of this behavior is that the inclusion of x^8 and higher order terms of the Taylor series truncation influences only those elements of the L matrix beyond the 20×20 truncation limit. To include the effects of the x^8 truncation of the Taylor Series, larger truncations (30×30) of the L matrix must be made.

In the next chapter a two dimensional problem will be examined. The problem chosen for this investigation is the Kepler problem. Questions that we hope to investigate include the behavior of the approximation technique on a two dimensional problem, what is the best basis to use, and how close the approximation results come to approximating the exact solution.

CHAPTER 6

APPLICATION OF APPROXIMATION TECHNIQUE TO KEPLER PROBLEM

In order to further test the approximation scheme on a two-dimensional problem, the Kepler problem for motion in a central inverse r squared force in the plane is studied. Using cartesian coordinates, the Hamiltonian is written as

$$H(x, y, p_x, p_y) = \frac{1}{2m}\left\{(p_x)^2 + (p_y)^2\right\} - \frac{\mu m}{\sqrt{x^2 + y^2}}, \tag{6.1}$$

where $\mu = GM$, G being the universal gravitational constant and M the mass of the earth, the gravitational potential being described by the usual radial function

$$V(r) = -\frac{\mu}{r} .$$

The same Hamiltonian expressed in terms of cylindrical polar coordinates (r, γ) is

$$H(r, \gamma, p_r, p_\gamma) = \frac{1}{2m}\left\{(p_r)^2 + \frac{1}{r^2}(p_\gamma)^2\right\} - \frac{\mu m}{r} , \tag{6.2}$$

where γ is the azimuthal angle measured from the x-axis. Since γ is an ignorable coordinate, the canonical momenta p_γ is constant and will be denoted by l_γ. The goal of this chapter is the application of polynomial basis vectors to

this non-linear problem in an analogous manner to the technique of previous chapters. Initially we start from some simple solution for the motion and study deviations from this simple motion. The simplest motion is uniform circular motion. We will study the deviations from pure circular motion. These deviations are expected to be similar to those deviations produced when the trajectories resemble epicycles.

To commence the study, a variable, q_r, defined as the fractional radial deviation of the radial component from some the equilbrium value, denoted by r_o is introduced. Then r is expressed as

$$r = r_o(1 + q_r) \ , \qquad (6.3)$$

so that the Hamiltonian can be written

$$H(q_r, \gamma, p_r, l_\gamma) = \frac{1}{2m}\left\{\frac{(p_r)^2}{r_o^2} + \frac{(l_\gamma)^2}{r_o^2(1 + q_r)^2}\right\} - \frac{\mu m}{r_o(1 + q_r)} \ . \qquad (6.4)$$

To facilitate subsequent numerical calculations, this Hamiltonian is converted to reflect a canonical set of units. This amounts to a rescaling of the canonical coordinates by taking the gravitational constant times the mass of the earth as 1 ($=\mu$), while taking the distance unit as the radius of the earth. This then forces a redefinition of the time scale. Let this new time be

denoted as τ. Then in this rescaled set of coordinates, the velocity is given by

$$\frac{d\vec{r}}{d\tau} = r_o \frac{dq_r}{d\tau} \hat{e}_r + r_o(1 + q_r) \frac{d\gamma}{d\tau} \hat{e}_\gamma ,$$

where $\hat{e}_r$ and $\hat{e}_\gamma$ are unit vectors in the radial and azimuthal directions respectively, so that the new momenta are given by

$$p_r' = (r_o)^2 \left[\frac{dq_r}{d\tau}\right] \quad ; \quad l_\gamma' = (r_o)^2(1 + q_r)^2 \left[\frac{d\gamma}{d\tau}\right] . \tag{6.5}$$

The Hamiltonian in canonical variables, denoted by $\mathcal{H}$, is then written

$$\mathcal{H}(q_r, \gamma, p_r', l_\gamma') = \frac{1}{2}\left\{\frac{(p_r')^2}{(r_o)^2} + \frac{(l_\gamma')^2}{r_o^2(1 + q_r)^2}\right\} - \frac{1}{r_o(1 + q_r)} , \tag{6.6}$$

where r_o is measured in units of earth radii and q_r is dimensionless. Keeping the radial deviation from equilbrium q_r small and expanding $\mathcal{H}$ to third order in q_r, one can obtain

$$\mathcal{H}(q_r, \gamma, p_r', l_\gamma') = \frac{1}{2(r_o)^2}(p_r')^2 + \frac{(l_\gamma')^2}{2(r_o)^2}\left[1 - 2q_r + 3q_r^2 - 4q_r^3\right] - \frac{1}{r_o}\left[1 - q_r + q_r^2 - q_r^3\right] . \tag{6.7}$$

The equilbrium value, r_o, can be determined by requiring

that the coefficient of the term linear in q_r vanish. This is equivalent to requiring that the equilibrium radial force vanish, which produces the equilibrium condition

$$r_o = (l_\gamma')^2 \quad . \tag{6.8}$$

Using the coordinate-momenta pair q_r, p_r' and γ, l_γ', the Liouville operator is defined in the usual way. Since

$$\mathcal{L}(l_\gamma') = [l_\gamma', H] = 0$$

l_γ' is a constant, and an initial set of basis vectors is chosen as γ, q_r, and p_r'. The action of the Liouville operator on this initial basis yields

$$\begin{aligned}
\mathcal{L}(\gamma) &= \frac{1}{(r_o)^2}(l_\gamma')\left[1 - 2q_r + 3q_r^2 - 4q_r^3\right] , \\
\mathcal{L}(q_r) &= \frac{1}{(r_o)^2}(p_r') , \\
\mathcal{L}(p_r') &= \left[\frac{1}{(r_o)^2}(l_\gamma')^2 - \frac{1}{r_o}\right] \\
&\quad + \left[-\frac{3}{(r_o)^2}(l_\gamma')^2 + \frac{2}{r_o}\right]q_r \\
&\quad + \left[\frac{6}{(r_o)^2}(l_\gamma')^2 - \frac{3}{r_o}\right]q_r^2 ,
\end{aligned} \tag{6.9}$$

Since $\mathcal{L}(\gamma)$ gives the constant $l_\gamma'/(r_o)^2$, the constant "vector" 1 is added to the initial basis. Under this augmented basis, the smallest L matrix that can be constructed is a 4×4, exhibited below:

$$
\begin{array}{c|cccc}
 & 1 & r & q_r & p_r' \\
\hline
1 & 0 & 0 & 0 & 0 \\
r & \dfrac{l_r'}{(r_o)^2} & 0 & \dfrac{-2\,l_r'}{(r_o)^2} & 0 \\
q_r & 0 & 0 & 0 & \dfrac{1}{(r_o)^2} \\
p_r' & \dfrac{(l_r')^2}{(r_o)^2} - \dfrac{1}{r_o} & 0 & -\dfrac{3}{(r_o)^2}(l_r')^2 + \dfrac{2}{r_o} & 0
\end{array}
= L
$$

Invoking the equilibrium relationship (equation 6.8), the above matrix is recognized as an upper Hessenberg matrix when written in terms of the constant r_o,

$$
\begin{array}{c|cccc}
 & 1 & r & q_r & p_r' \\
\hline
1 & 0 & 0 & 0 & 0 \\
r & (r_o)^{-3/2} & 0 & -2(r_o)^{-3/2} & 0 \\
q_r & 0 & 0 & 0 & (r_o)^{-2} \\
p_r' & 0 & 0 & -(r_o)^{-1} & 0
\end{array}
= L \, . \qquad (6.10)
$$

The eigenvalues of this matrix are 0 (multiplicity 2), and $\pm i(r_o)^{-3/2}$.

With these eigenvalues the r(t) functions are found as

$$r_1(t) = 1$$

$$r_2(t) = t$$

$$r_3(t) = \frac{1}{\omega^2}\left[1 - \cos(\omega t)\right] + \frac{i}{\omega^2}\left[\omega t - \sin(\omega t)\right]$$

$$r_4(t) = \frac{1}{\omega^3}\left[\omega t - \sin(\omega t)\right],$$

where ω has been used for $(r_o)^{-3/2}$. The P matrices are

$P_o = I$, $P_1 = L$,

$$P_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2(r_o)^{-7/2} \\ 0 & 0 & -(r_o)^{-3} & 0 \\ 0 & 0 & 0 & -(r_o)^{-3} \end{bmatrix},$$

and

$$P_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2(r_o)^{-9/2} & 2i(r_o)^{-5} \\ 0 & 0 & i(r_o)^{-9/2} & -(r_o)^{-5} \\ 0 & 0 & (r_o)^{-4} & i(r_o)^{-9/2} \end{bmatrix}.$$

Applying the Putzer algorithm, the matrix for e^{Lt} is

$$e^{Lt} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \omega t & 1 & -2\sin(\omega t) & \frac{-2}{\omega(r_o)^2}\left[1 - \cos(\omega t)\right] \\ 0 & 0 & \cos(\omega t) & \frac{1}{\omega(r_o)^2}\sin(\omega t) \\ 0 & 0 & -\omega(r_o)^2\sin(\omega t) & \cos(\omega t) \end{bmatrix}, \tag{6.11}$$

where $\omega = (r_o)^{-3/2}$, is the magnitude of the imaginary part of the complex eigenvalue. The time dependence of the coordinates r, q_r, and p_r' are obtainable from the matrix multiplication

$$\vec{u}(t) = e^{Lt}\,\vec{u}(0) ,$$

where the vectors $\vec{u}(t)$ and $\vec{u}(0)$ are defined as

$$\vec{u}(t) = \begin{bmatrix} 1 \\ r(t) \\ q_r(t) \\ p_r'(t) \end{bmatrix} \quad \text{and} \quad \vec{u}(0) = \begin{bmatrix} 1 \\ r_o \\ (q_r)_o \\ (p_r')_o \end{bmatrix} . \tag{6.12}$$

The time dependence of the coordinates are given by the functions

$$r(t) = r_o + \omega t - 2(q_r)_o\sin(\omega t) - 2(p_r')_o\frac{(r_o)^2}{\omega}\left[1 - \cos(\omega t)\right]$$

$$q_r(t) = (q_r)_o\cos(\omega t) + (p_r')_o\frac{1}{\omega(r_o)^2}\sin(\omega t) ,$$

$$p_r'(t) = -(q_r)_o\,\omega(r_o)^2 \sin(\omega t) + (p_r')_o \cos(\omega t) \quad . \tag{6.13}$$

Since the L matrix obtained above contains zeros in the first row due to the inclusion of 1 as one of the initial basis vectors, the first row and first column of the above matrix and all higher truncations of the L matrix can be dropped provided the term $((r_o)^{-3/2})t$ is added to the resultant expression for $\gamma(t)$. This makes eigenvalue calculation easier for the higher truncations. The highest order truncation that will be discussed here is the 10×10 truncation.

Choosing the extended polynomial basis of

$$\gamma \quad \text{plus} \quad \left[q_r + p_r' \right]^n , \tag{6.14}$$

where n is an integer, whose highest value is related to the number of terms that one wishes to carry in the matrix truncation. For the 10×10 truncation of the L matrix, n is 3. The L matrix constructed from these basis vectors for the 10×10 truncation can be partitioned into the following block form

$$L = \begin{array}{c} \left[\begin{array}{c|c|c} \mathbf{A} & \mathbf{A} & \mathbf{A} \\ \hline 0 & \mathbf{B} & \mathbf{B} \\ \hline 0 & 0 & \mathbf{C} \end{array}\right] \begin{array}{l} \leftarrow 3 \\ \leftarrow 6 \\ \leftarrow 10 \end{array} \\ \begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ 3 & 6 & 10 \end{array} \end{array} , \tag{6.15}$$

where the boldface entries are blocks of varying dimensions

that are determined by the partitioning of the basis vectors used. The various blocks of the partitioned L matrix are defined in terms of canonical variables as follows:

$$\mathbf{A} = \begin{bmatrix} 0 & -2r_o^{-3/2} & 0 \\ 0 & 0 & r_o^{-2} \\ 0 & -(r_o)^{-1} & 0 \end{bmatrix},$$

$$\mathcal{A} = \begin{bmatrix} 3(r_o)^{-3/2} & 0 & 0 \\ 0 & 0 & 0 \\ 3(r_o)^{-1} & 0 & 0 \end{bmatrix},$$

$$\mathfrak{A} = \begin{bmatrix} -4(r_o)^{-3/2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 0 & 2(r_o)^{-2} & 0 \\ -(r_o)^{-1} & 0 & (r_o)^{-2} \\ 0 & -2(r_o)^{-1} & 0 \end{bmatrix},$$

$$\mathcal{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 3(r_o)^{-1} & 0 & 0 & 0 \\ 0 & 6(r_o)^{-1} & 0 & 0 \end{bmatrix},$$

and

$$C = \begin{bmatrix} 0 & 3(r_o)^{-2} & 0 & 0 \\ -(r_o)^{-1} & 0 & 2(r_o)^{-2} & 0 \\ 0 & -2(r_o)^{-1} & 0 & (r_o)^{-2} \\ 0 & 0 & -3(r_o)^{-1} & 0 \end{bmatrix} .$$

The eigenvalue structure of the partitioned 10×10 truncated L matrix is now examined. Let the modulus of the imaginary part of the smallest magnitude eigenvalue be denoted by β. For example, for an r_o value of 1.1, $(r_o)^{-3/2}$ is $(1.1)^{-3/2}$ = .8667841365D+00. For the 10×10 truncation of the L matrix, the eigenvalue spectrum, denoted by $\sigma(\lambda)$, has the structural form shown in Table 6.1 below.

A series of plots arranged by increasing truncation size of the L matrix are presented next. The exact results referred to in the plots were produced by the Runge-Kutta-Gill (RKG) numerical integrator. Figures 6.1 to 6.4 represent the results of the approximation scheme for the 3×3 truncation. Figures 6.5 to 6.8 contain the results of the approximation scheme for the 6×6 truncation while Figures 6.9 to 6.12 incorporate the results of the approximation scheme for the 10×10 truncation of the L matrix.

Figure 6.1 contains a comparative plot of the azimuthal angle, γ, calculated by RKG integration (solid

Table 6.1

Eigenvalue Spectrum for Two-Dimensional Kepler Problem

$r_o = 1.1$, $\beta = (r_o)^{-3/2} = .866784D+00$

partition number	dim	eigenvalue	
		real part	imag part
1	3	0	$\pm 1\beta$, 0
2	6	0	$\pm 2\beta$, 0
3	10	0	$\pm 3\beta$, $\pm\beta$

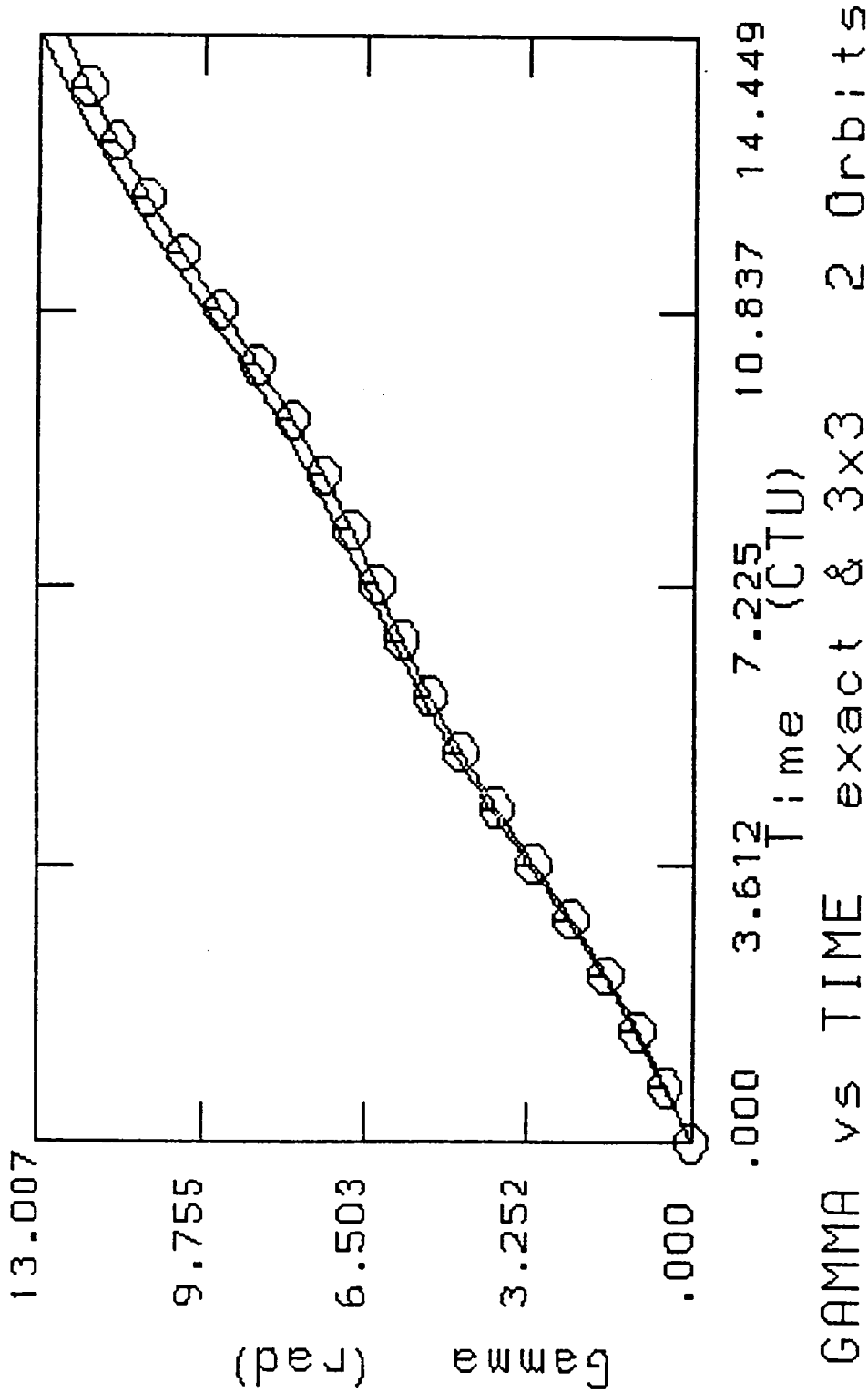
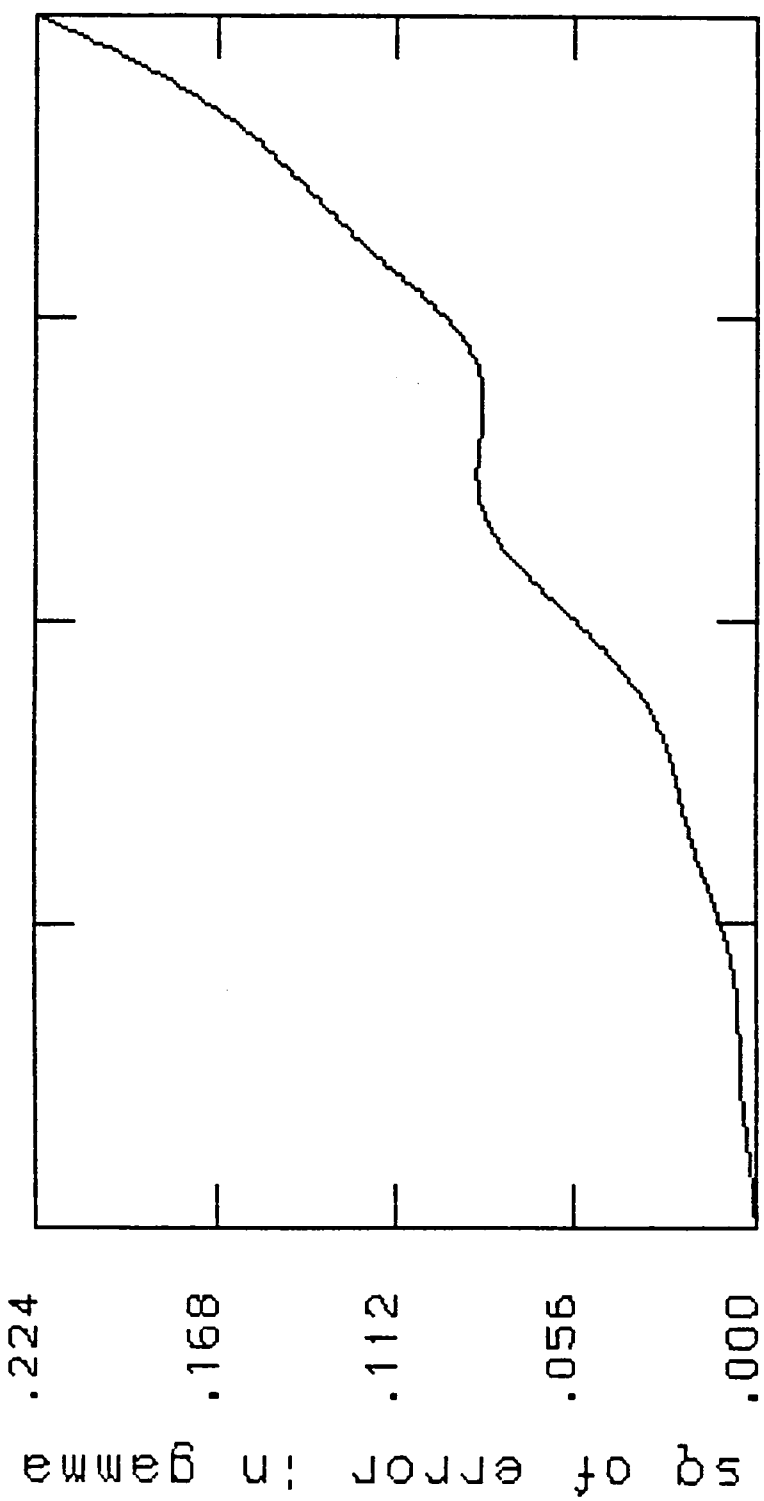

13.007
9.755
6.503
3.252
.000
Gamma (rad)
.000
3.612
7.225
10.837
14.449
Time (CTU)
GAMMA vs TIME exact & 3x3 2 Orbits

Figure 6.1

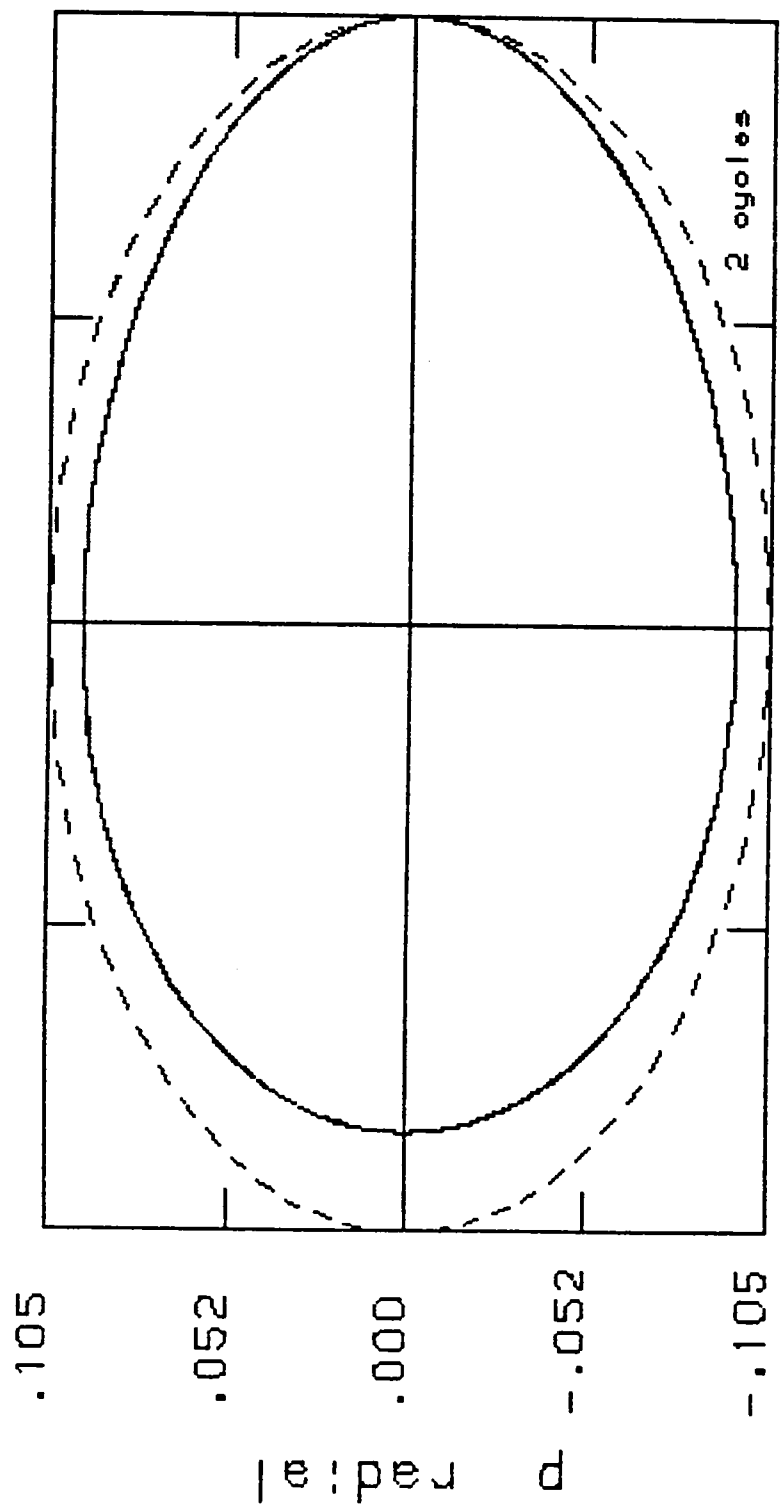
line) and by the approximation scheme (line with plotting symbol) using the 3×3 L matrix truncation. Data are presented for 2 orbits. The approximated azimuthal angle lags the exact azimuthal angle. Secular terms have not been removed from either data set. Figure 6.2 exhibits a plot of the square of the difference between the exact solution and the approximated solution for the azimuthal angle (γ) as a function of time. Both the oscillatory nature and the secular nature of the errors are evident. Figure 6.3 depicts a phase plot of the radial variables for both the exact (RKG) solution (solid line) and the approximation scheme (dashed line) using the Putzer formula for the 3×3 truncation. Two cycles of data are represented.

The phase plot of the exact data appears as egg-shaped while the phase plot of the approximation scheme is more elliptical in nature. This graph is not what one might expect of the exact situation. However, in Figure 6.7 we see that the 6×6 truncation moves closer to this shape. It is not immediately apparent why this behavior is exhibited. A phase plot of the differences between the exact and approximated radial position and momentum of the data of Figure 6.3 is presented in Figure 6.4. In the difference phase plot the behavior of the data is noted as spiraling outward and somewhat off center. Figure 6.5 is the 6×6



SQ of ERROR in GAMMA vs TIME for 3x3

Figure 6.2



PHASE PLOT exact & 3x3 approx q0=.1

Figure 6.3

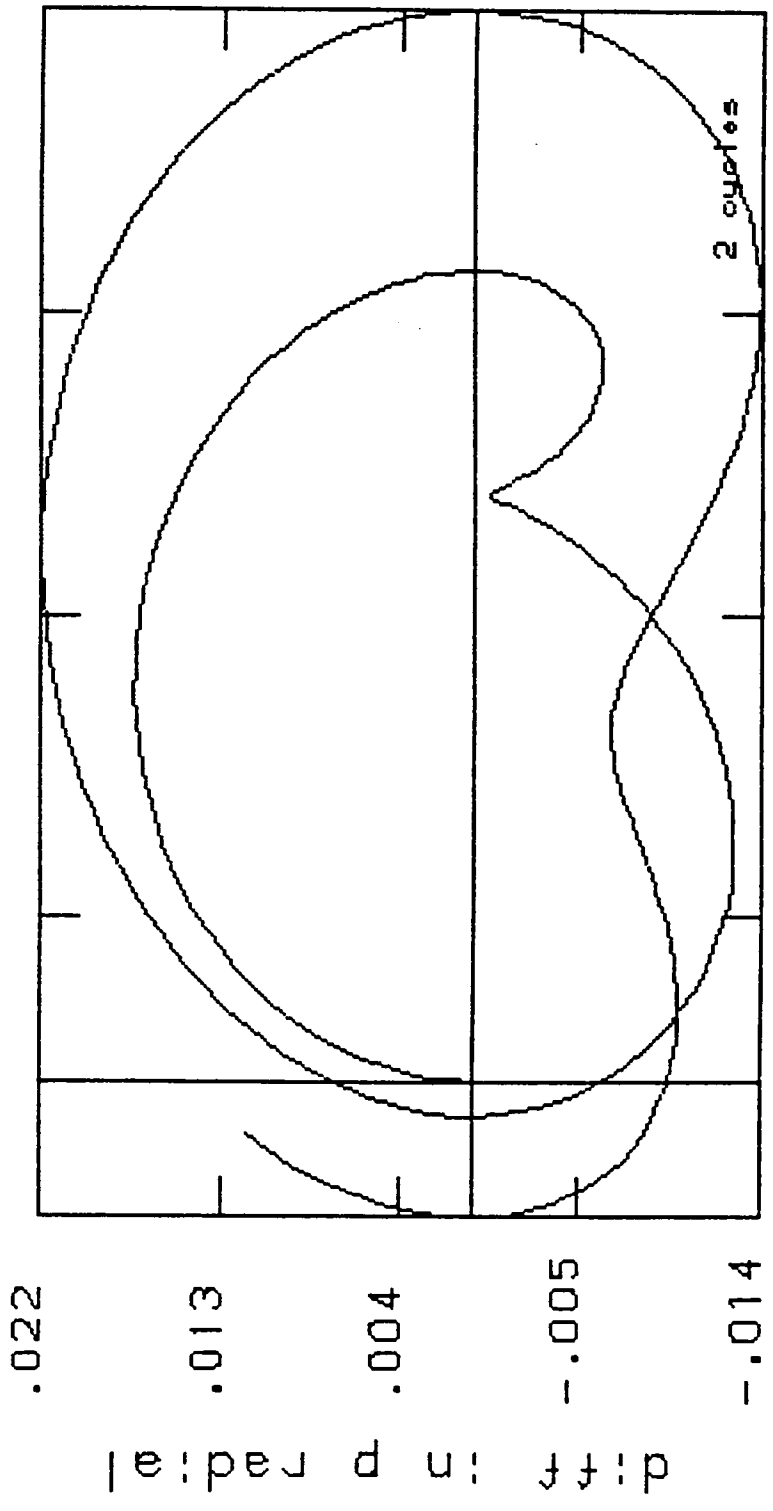

PHASE DIFF exact & 3x3 approx q0=.1
diff in q radial
diff in p radial
2 cycles
-.004
.004
.012
.020
.028
.022
.013
.004
-.005
-.014

Figure 6.4

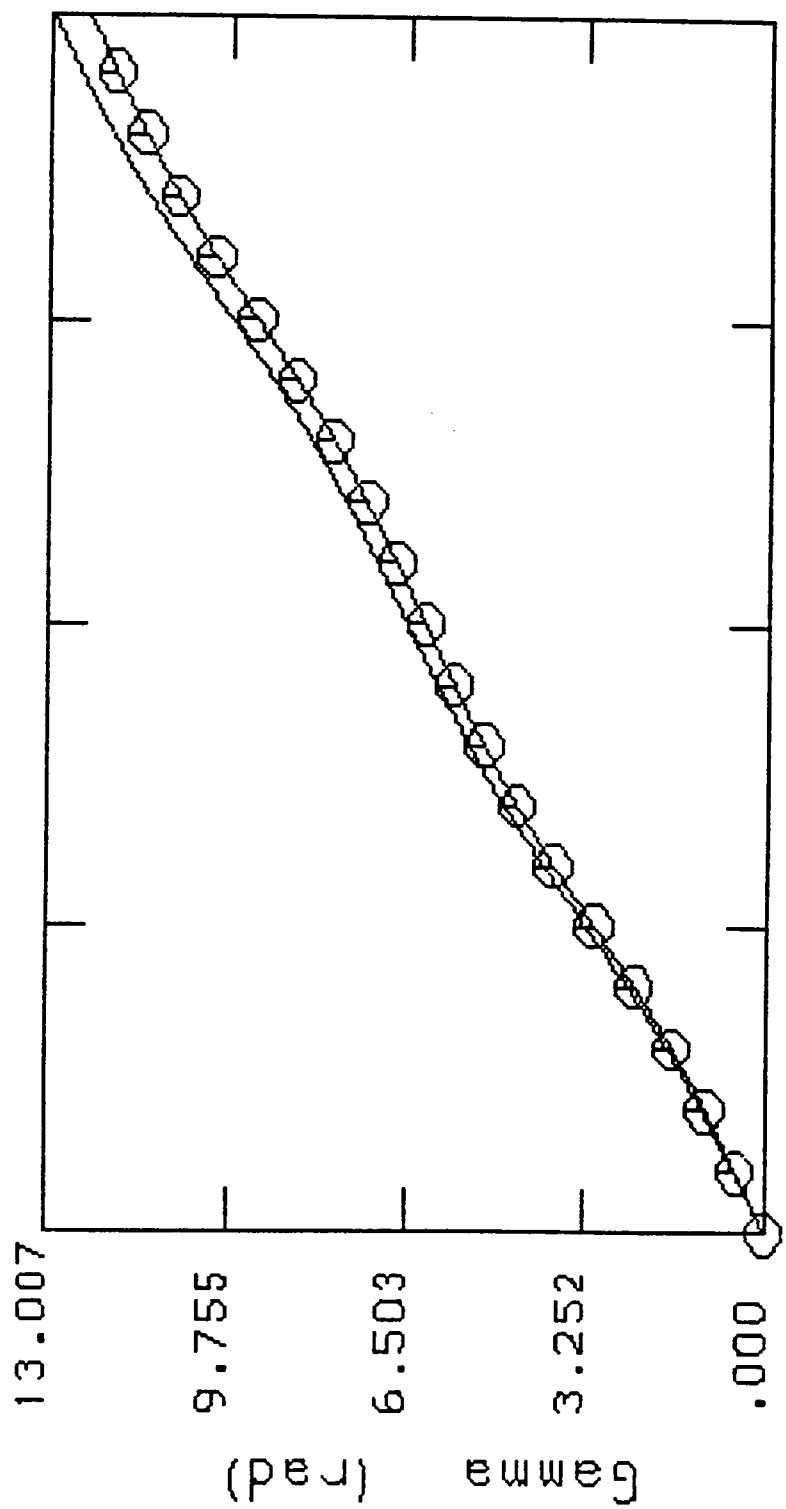

13.007
9.755
6.503
3.252
.000
Gamma (rad)
.000
3.612
7.225
10.837
14.449
Time (CTU)
GAMMA vs TIME exact & 6x6 2 Orbits

Figure 6.5

truncation analog of Figure 6.1 for 2 cycles of data. The azimuthal lag appears to have increased slightly. Secular and oscillatory behavior of the square of the error in the azimuthal angle γ are the principal features of Figure 6.6. Figure 6.7 is a phase space plot of the radial variables for the exact RKG solution (solid line) and the approximation scheme (dashed line) using the 6×6 L matrix truncation. Again two cycles are presented. In comparison to the previous plot for the 3×3 case, we observe that the approximation has moved closer to the exact solution of the egg-shaped curve. Figure 6.8 displays a phase plot of the differences between the exact and approximated radial positions and momenta for the data of Figure 6.7. As before, the plot of the differences in phase space opens spirally outward and is located off center, with some compression at the top of the plot.

Figure 6.9 is a comparitive plot of the azimuthal angle γ showing the exact (RKG) solution (solid line) and the approximated solution (denoted by the line containing the plotting symbol) for the 10×10 truncation. The azimuthal lag appears to have decreased slightly. Secular and oscillatory behavior of the square of the error in the azimuthal angle γ are the features of Figure 6.10. The magnitude of the error has decreased slightly, while the magnitude of the oscillatory component has increased.

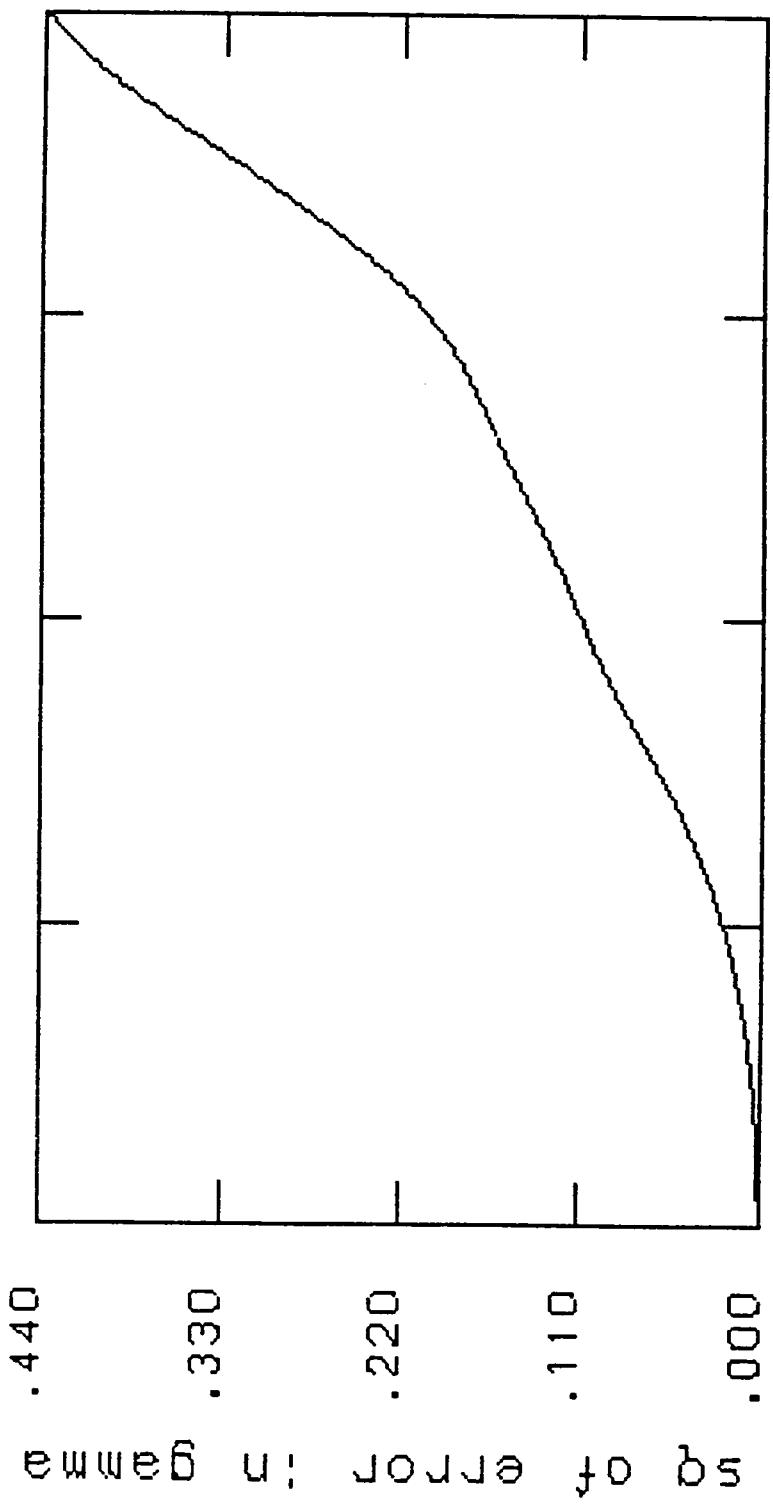

.440
.330
.220
.110
.000
sq of error in gamma
.000
3.612
7.225
10.837
14.449
Time (CTU)
SQ of ERROR in GAMA vs TIME for 6x6

Figure 6.6

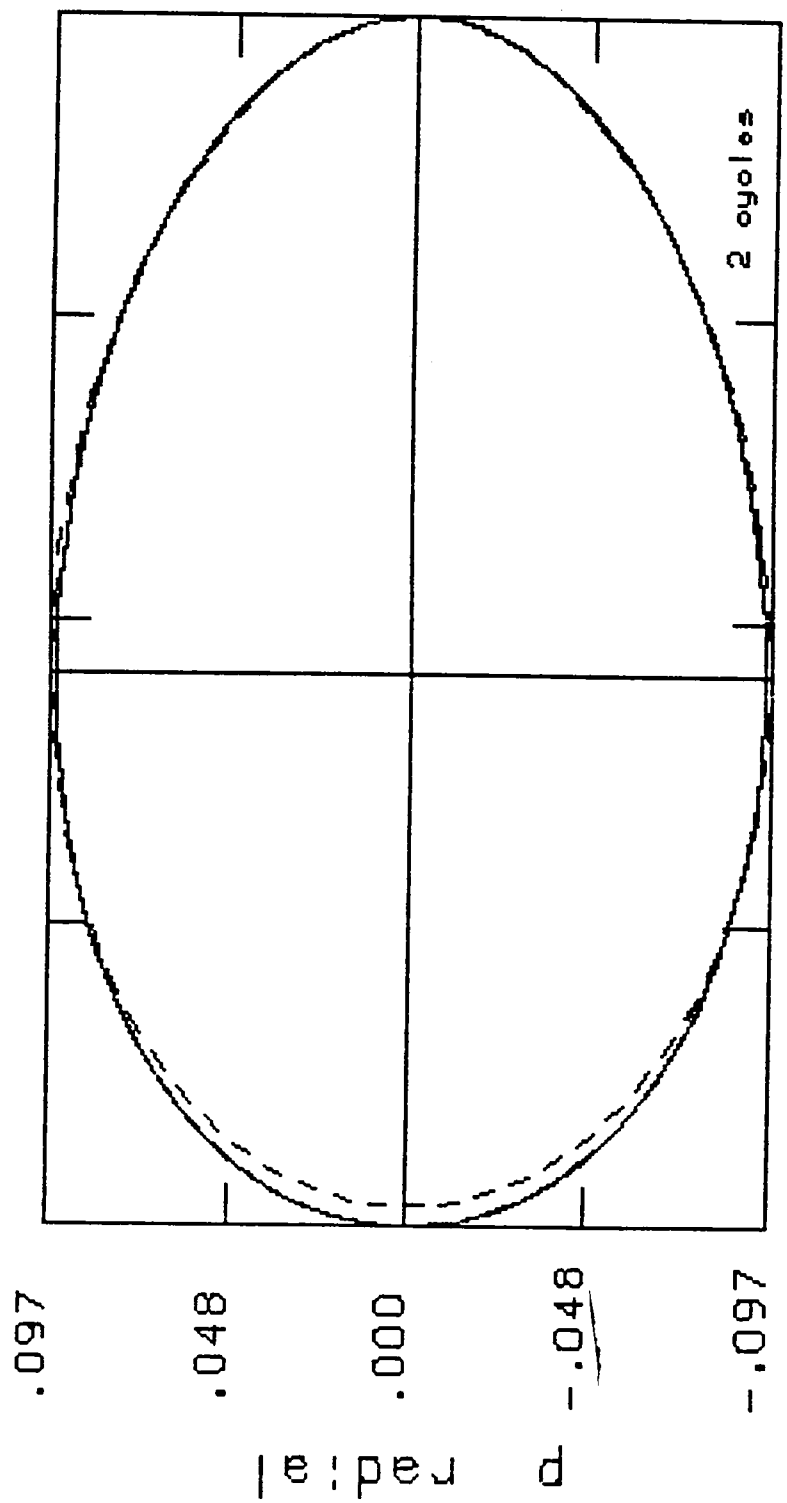

PHASE PLOT exact & 6x6 approx q0=.1
p radial
.097
.048
.000
-.048
-.097
-.083
-.037
.008
.054
.100
q radial
2 cycles

Figure 6.7

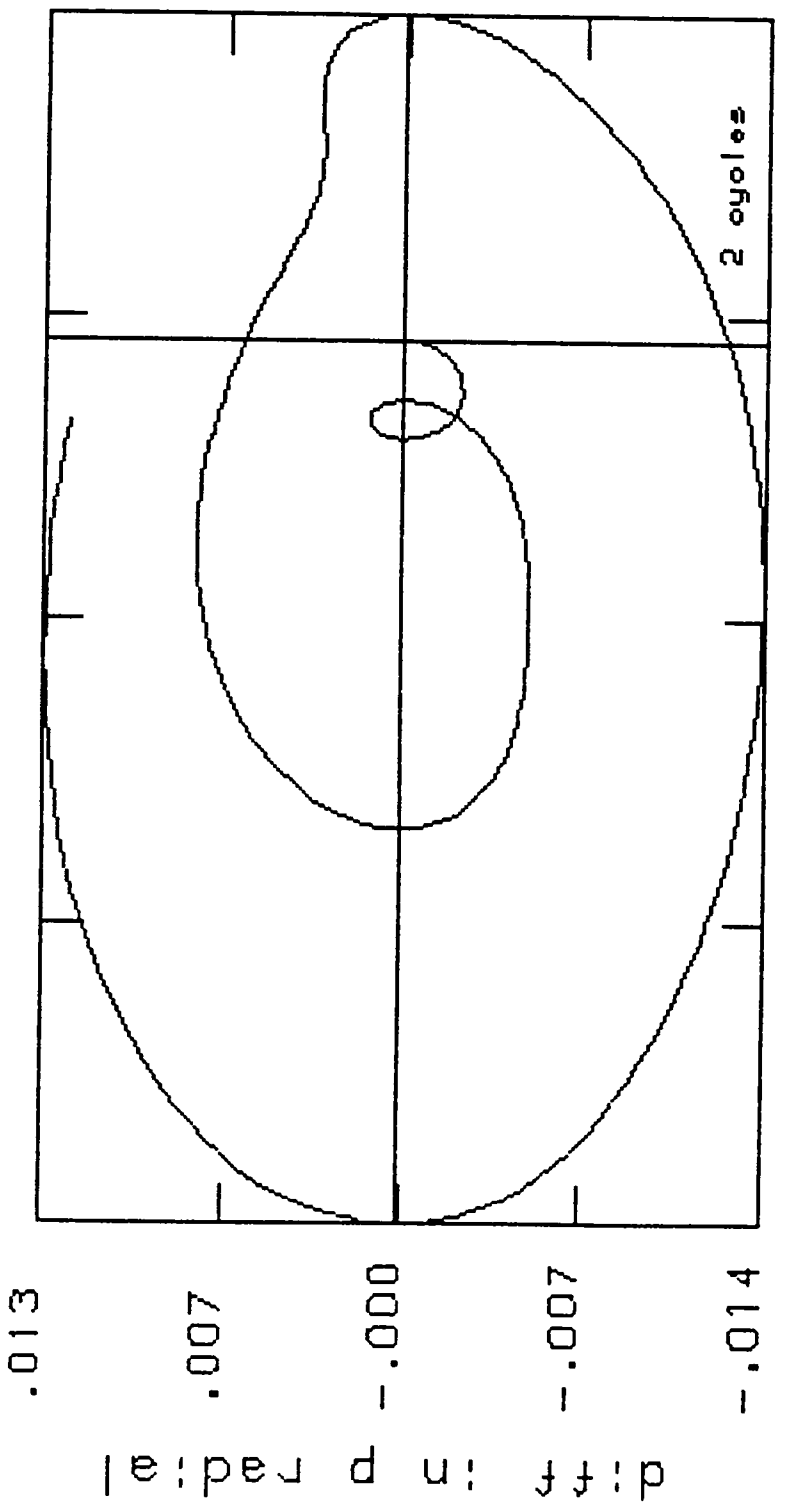
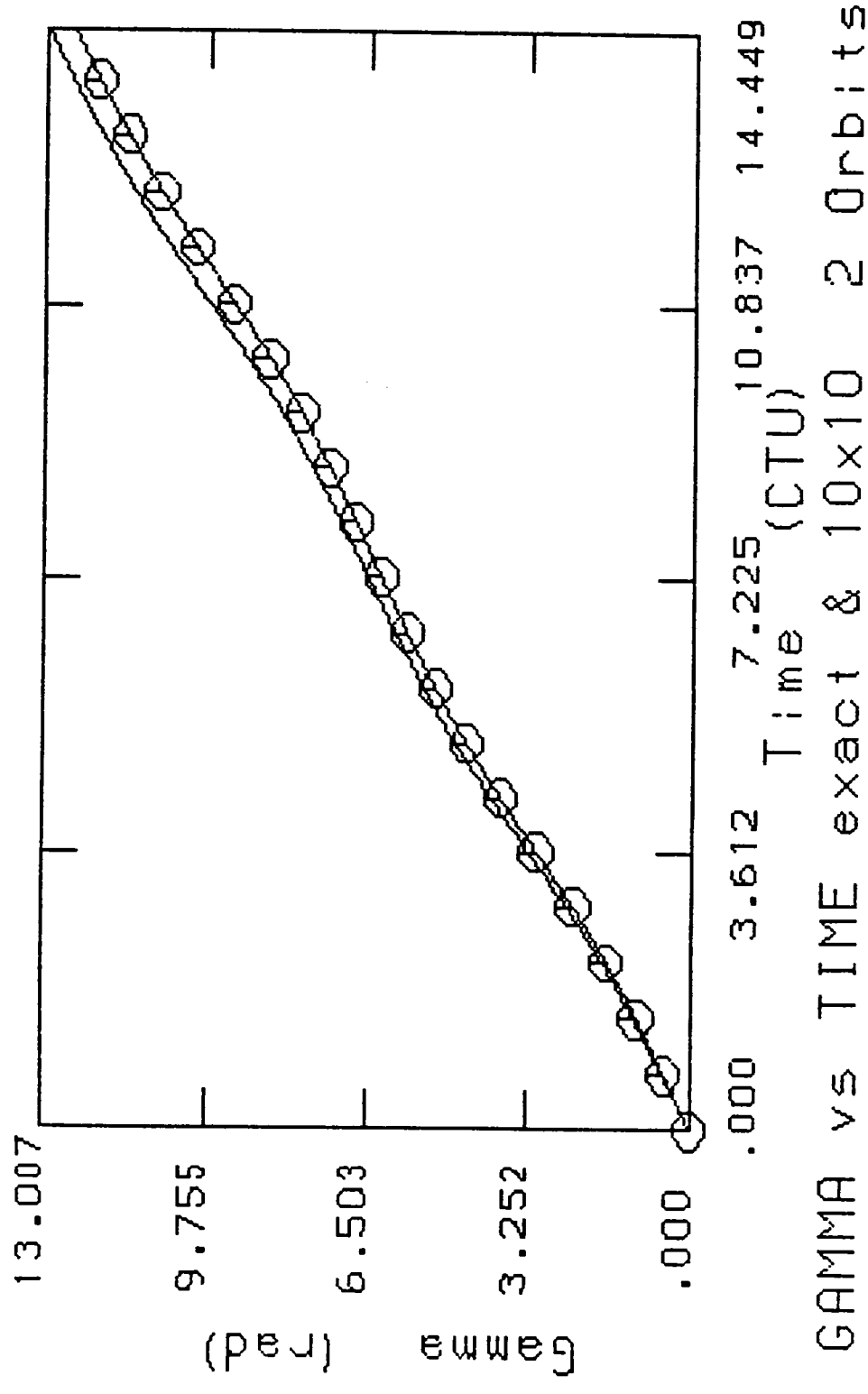
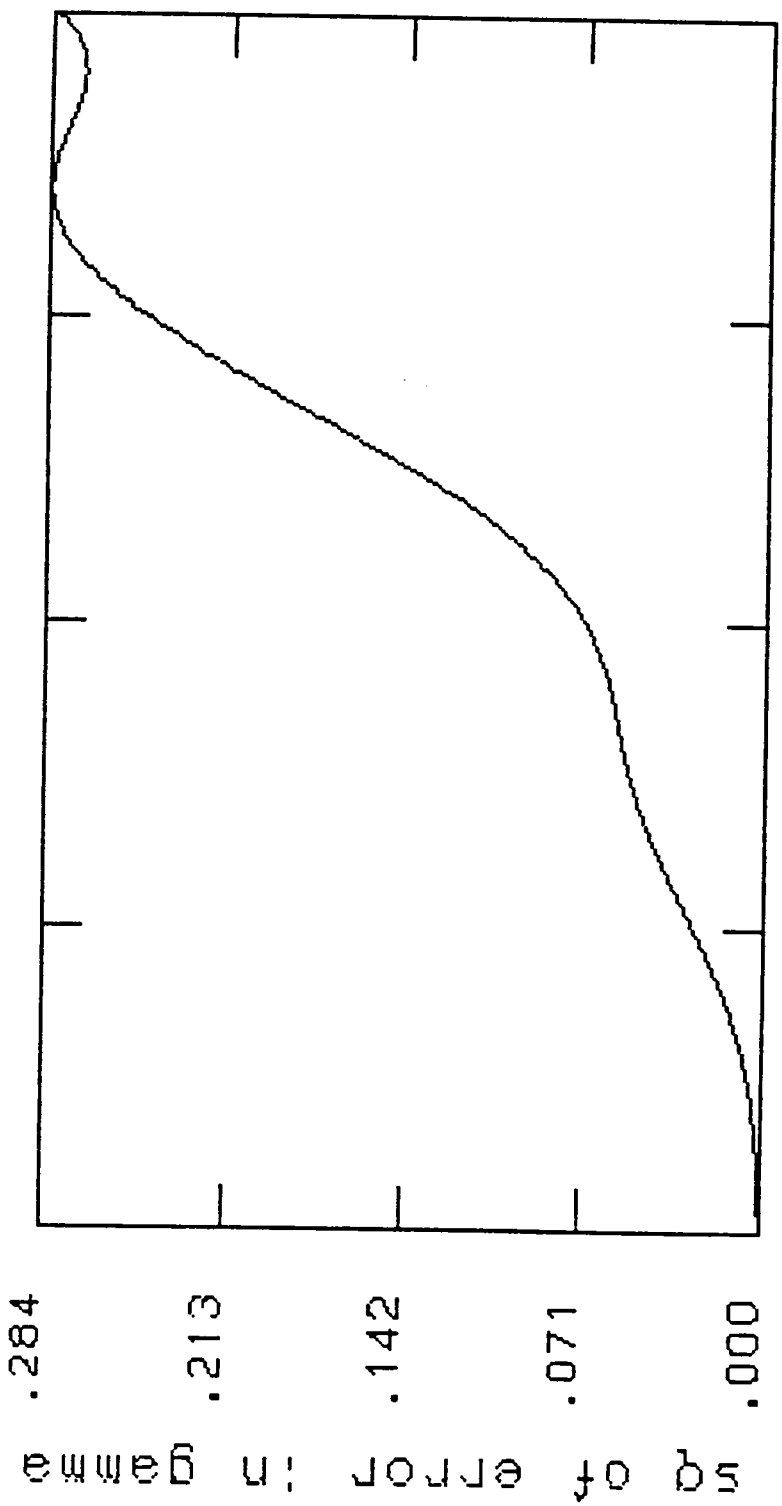

PHASE DIFF exact & 6x6 approx q0=.1
diff in q radial
diff in p radial
2 cycles
.013
.007
-.000
-.007
-.014
-.016
-.010
-.005
.000
.006

Figure 6.8



GAMMA vs TIME exact & 10x10 2 Orbits

Figure 6.9



SQ of ERROR in GAMMA vs TIME 10x10

Figure 6.10

Figure 6.11 displays a plot in phase space of the radial variables for the exact (RKG) solution (solid line) and the approximation scheme using the 10×10 L matrix truncation (dashed line). Two cycles of data are shown. In comparison with the plot for the 6×6 case, the dashed curve appears to be moving away from the solid line egg-shaped curve representing the exact data. This behavior is similar to the behavior of the approximated solutions of the 30×30 and higher truncations of the L matrix for the perturbed simple harmonic oscillator studied in Chapter 3. For both cases higher order time dependent terms are a factor when the approximation scheme is applied using higher order truncations. Figure 6.12 contains a plot in phase space of the differences between the exact and approximated solutions shown in Figure 6.11. The phase plot curve of the differences opens spirally outward. Appendix G contains the FORTRAN source for the driver for the eigenvalue calculation of the L matrix for the Kepler problem using QR decomposition routines of the EISPACK guide. Appendix H provides a listing of the FORTRAN source for the approximation scheme employing the Putzer algorithm for the 6×6 truncation of the L matrix for the Kepler problem. Appendix I presents the FORTRAN source listing for the Runge-Kutta Gill numerical integrator used for the Kepler problem.

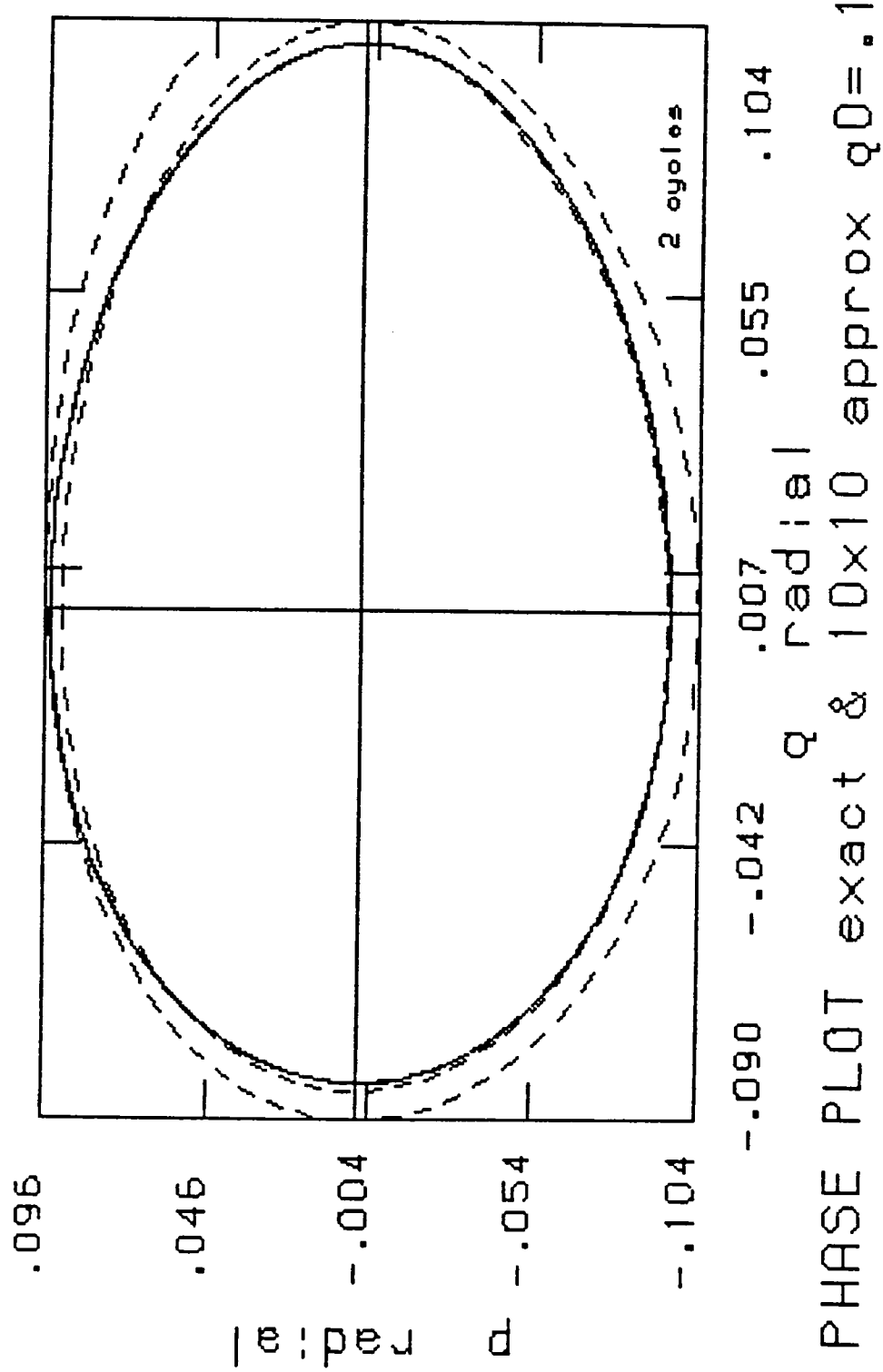

PHASE PLOT exact & 10x10 approx q0=.1
2 cycles
p radial
.096
.046
-.004
-.054
-.104
-.090
-.042
.007
.055
.104
q radial

Figure 6.11

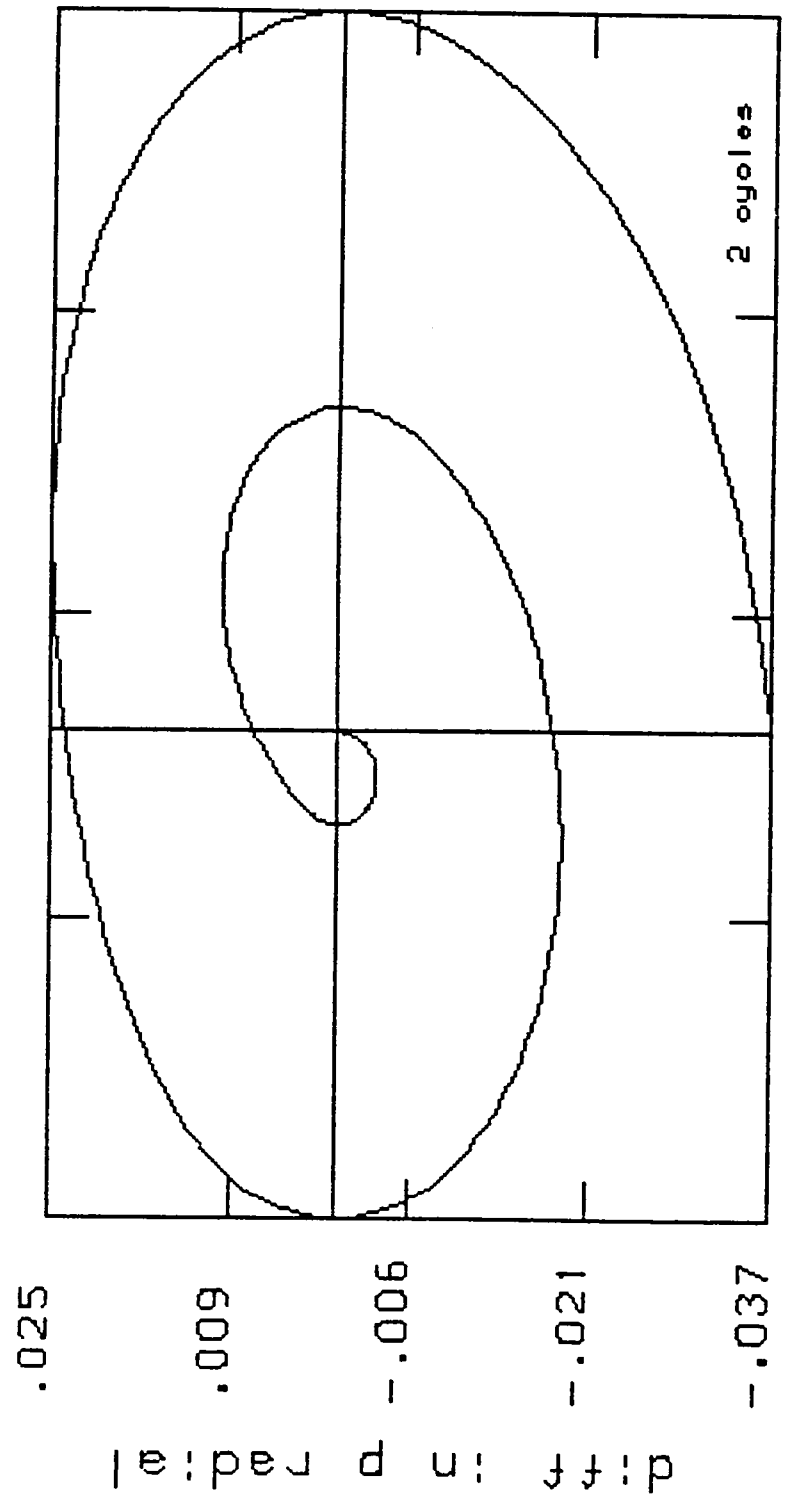

PHASE DIFF exact & 10x10 approx q0=.1
diff in q radial
diff in p radial
2 cycles
-.020
-.008
.005
.017
.030
.025
.009
-.006
-.021
-.037

Figure 6.12

Table 6.2 presents the statistics for the 3×3 truncation, while Table 6.3 performs the same function for the 6×6 truncation and Table 6.4 contains the statistics for the 10x10 truncation. An overall comparison of the statistics of the radial deviation and corresponding momentum for all truncations over two orbits can be obtained by examining Figure 6.13. The errors of the 3×3 truncation appear to be reduced by about one-third when compared to the errors of the 6×6 truncation. The errors associated with the 10×10 truncation are slightly greater than the errors of the 3×3 and appear to be twice the errors of the 6x6 truncation. A possible explaination for this behavior is that higher truncations tend to produce deteriorated results in much the same way that higher truncations of the perturbed simple harmonic oscillator studied in Chapter 3 gave poor results. The error in these statistics can be analyzed as coming from two separate sources. The first source of error is encountered by the truncation of the L matrix itself. The second source of error is identified with the truncation of the Taylor series expansions used in the Hamiltonian. There is no quantifable way to identify the error from each of these independent error sources.

Before generating data for larger truncations of the L matrix for the planar Kepler problem, the cause of the

Table 6.2

Statistics for $\Delta\gamma$, Δq, and Δp for exact solution and 3×3 approximation using Putzer algorithm with RKG integration

$r_o = 1.1$, $\beta = (r_o)^{-3/2} = .866784D+00$

number of orbits	1 σ values		
	azimuth angle	radial	
		position	momentum
2	.127799	.009075	.010470
6	.390188	.019464	.020576
10	.651213	.030875	.032147
16	1.041829	.047486	.049269

Table 6.3

Statistics for $\Delta\gamma$, Δq, and Δp for exact solution and 6×6 approximation using Putzer algorithm with RKG integration

$r_o = 1.1$, $\beta = (r_o)^{-3/2} = .866784D+00$

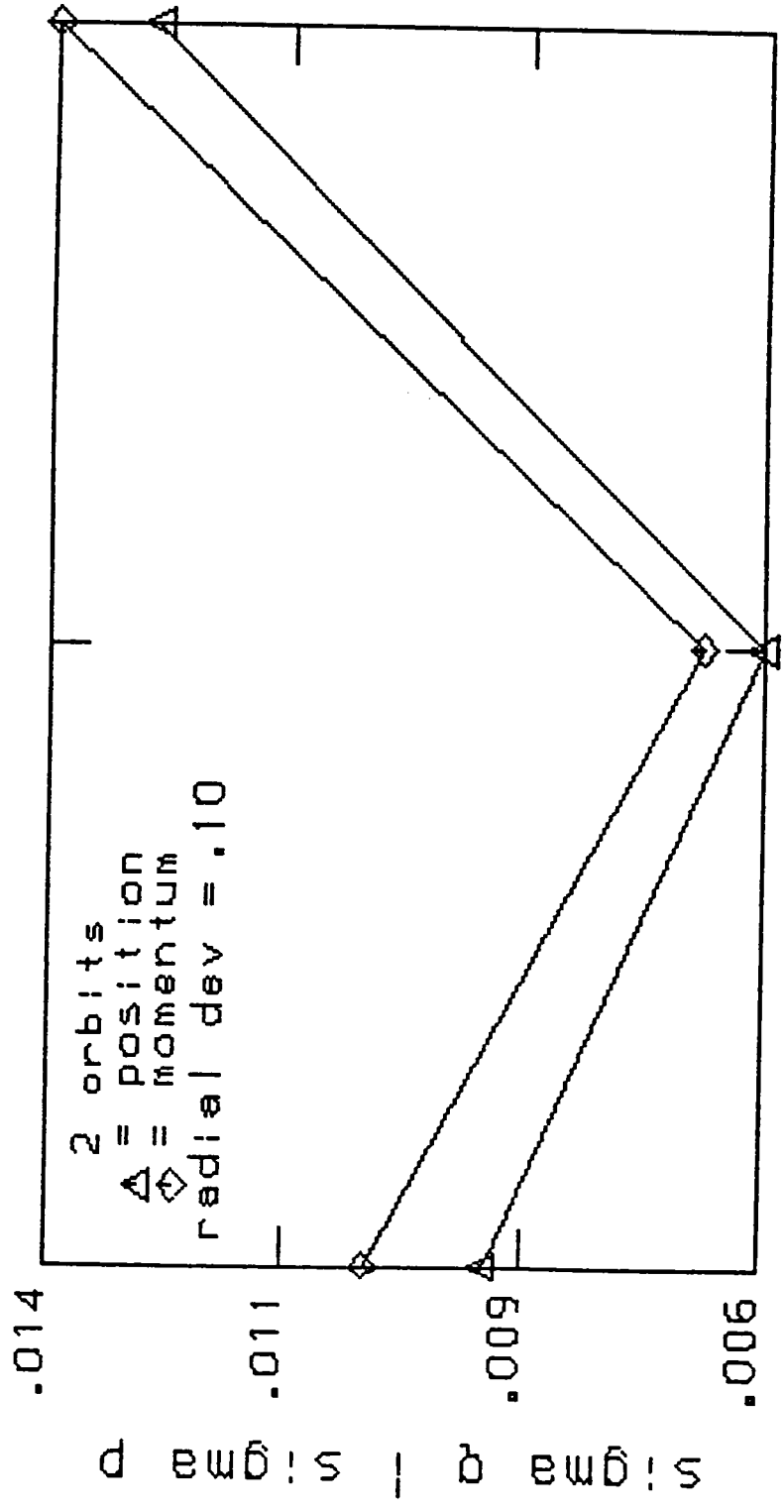
number of orbits	1 σ values		
	azimuth angle	radial position	radial momentum
2	.189285	.005823	.006521
6	.555482	.017438	.018299
10	.924154	.028757	.030023
16	1.476836	.044811	.046778

Table 6.4

Statistics for $\Delta\gamma$, Δq, and Δp for exact solution and 10×10 approximation using Putzer algorithm with RKG integration

$r_o = 1.1$, $\beta = (r_o)^{-3/2} = .866784D+00$

number of orbits	1 σ values		
	azimuth angle	radial position	radial momentum
2	.173770	.013097	.014257
6	.526421	.040604	.042921
10	.878537	.069211	.072958
16	1.406023	.115818	.122196



STATISTICS by Truncation Size: Kepler

Figure 6.13

divergence introduced by the approximation method when larger truncations are used needs to be definitively pinpointed and corrected. The secular terms introduced by the approximation which grow as t^n need to be studied, identified, and possibly removed. An analytical examination of how the approximation method employs the information introduced by the larger truncations should provide relevant key information to solve this problem.

The indication is that the approximation method will work on the Kepler problem. The success of the 6×6 truncation suggests that, with more work, we can hope to use the approximation technique in an analytic method for making large, but relatively accurate steps whose error is controlled and confined when solving the Kepler problem. Such a larger step size could possibly allow a faster method of calculating the location of satellites than the small step size of the iterative methods currently being used. The next chapter will outline areas and present ideas, and suggestions on how the research may be continued and extended. Collective comments will also be made on the results achieved by this research.

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS FOR EXTENDING AND CONTINUING THE WORK

The purpose of this chapter is to offer some conclusions about previous work and to indicate areas of this research that can be constructively extended. Before proceeding, however, a short comment should be made concerning the computer environment and the limitations under which this work was done. The work performed was implemented using a small microcomputer having a 16 bit wordsize. The intent was to study the "small" truncations involved in an approximation scheme, consequently a small computer could perform the study. Whether the choice of performing the work on a small microcomputer affected the results obtained is a legitimate issue that has not been permanently decided. For at least one problem involving a 20×20 matrix representation for the Liouville operator (see Chapter 3), preliminary evidence indicates that this choice of using the microcomputer did not adversely affect the results obtained. For this truncation, the same data were produced by both the small microcomputer and a DEC MicroVAX computer. The standard deviation of the differences of the position calculated by the two computers was .000340. The corresponding figure for the differences in the momentum was .001481.

We now take the opportunity to reflect on the results

obtained. For each of the non-trival problems examined, the approximation scheme produced good results for the short time regime (2-3 cycles). Beyond the short time regime, the values obtained for the position and the momentum deteriorated. The source of this deterioration is the truncation approximation in which the exponential of the Liouville operator for a finite matrix truncation. This approximation shares with the simplest Taylor series approximation to the motion the property that convergence is only for short times. The approximations explored here are not just power series in t, but oscillating factors multiplying such a series. As seen in previous chapters increasing the size of the truncation does improve the approximation for short times. The fact that we get good agreement for 2-3 cycles is quite an accomplishment for this method in comparison to other short time approximations.

There are several areas that are natural extensions of the previous work which have not been considered heretofore. Efforts can be divided into four different, independent areas. Results from some areas have the capacity of producing a beneficial effect on the other areas. Each of these areas is listed and discussed in turn. The first area is the further examination of how the approximation method handles motion, first in two

dimensions, and then in three dimensions to extend the results obtained in Chapter 6. The second area involves the extension of the Putzer algorithm to non-constant matrices. The third area is the identification of the complete time dependence of the solution produced by the approximation method in solving the perturbed simple harmonic oscillator studied in Chapter 3. A fourth area is the formulation of implementation details enabling the methodology to solve a practical real-life problem.

The first area is the further exploration of how the approximation scheme treats motion in two dimensions. The effects that multiple constants of motion have on the selection of the set of basis vectors employed by the approximation scheme need to be better understood. As seen in Chapter 6, since the azimuthal momentum coordinate was a constant, a constant vector had to be included in the initial basis set. An L matrix was obtained relative to this basis.

One test to ascertain the proper method for handling multiple constants of the motion can be proposed. This test consists of the comparison of the results produced from the following tasks. The scheme should be applied to the unperturbed two-dimensional harmonic oscillator Hamiltonian

$$H(x, y, p_x, p_y) = \frac{1}{2m}\left\{ p_x^2 + p_y^2 \right\} + \frac{k}{2}\left\{ x^2 + y^2 \right\}$$

in cartesian coordinates. An analytic solution can be found. Then the scheme should be applied to the two-dimensional analog of the Hamiltonian of Chapter 3, e.g.

$$H(x, y, p_x, p_y) = \frac{1}{2m}\left\{ p_x^2 + p_y^2 \right\} + \frac{k}{2}\left\{ x^2 + y^2 \right\} + \frac{g}{4!}\left\{ x^4 + y^4 + 2x^2y^2 \right\}.$$

Next, the scheme should be applied to the equivalent of the first Hamiltonian expressed in polar coordinates, e.g. the approximation scheme should be applied to

$$H(r, \gamma, p_r, p_\gamma) = \frac{1}{2m}\left\{ p_r^2 + \frac{1}{r^2} p_\gamma^2 \right\} + \frac{k}{2} r^2 ,$$

and analytical results obtained. The analytical results provide a reference point for the calculations of the perturbed solutions. Next the scheme could be applied to the perturbed two-dimensional harmonic oscillator whose Hamiltonian in polar coordinates is

$$H(r, \gamma, p_r, p_\gamma) = \frac{1}{2m}\left\{ p_r^2 + \frac{1}{r^2} p_\gamma^2 \right\} + \frac{k}{2} r^2 + \frac{g}{4!} r^4 .$$

By making careful observations of the differences between

the results of these two Hamiltonians, better insight can be obtained. In the above polar coordinate cases, r^{-1} can be expanded in the manner of Chapter 6, e.g.

$$r^{-1} = \frac{1}{r_o(1 + q_o)} = (r_o)^{-1}\left[1 - q_o + q_o^2 - q_o^3 + \ldots \right].$$

The next application of the approximation procedure is to consider the celestial mechanical problem of the three dimensional perturbed Kepler problem. This is a natural extension of the work of the previous chapter. If a way can be found to make the approximation scheme work more effectively on a Kepler problem, a very valuable and practical methodology will ensue. The first task is finding a suitable set of coordinates and momenta to use. It is expected that the set of coordinates and momenta should be canonical to each other. Since most of the work done in celestial mechanics uses the Delaunay set of variables, this set should be tried as a natural first choice. Use of this set would facilitate the practical connections of any approximation scheme to be developed. The majority of the work involves the proper conversion of the disturbing function which is typically added to the potential into the chosen set of coordinates and momenta. Following the technique of the last chapter, the disturbing function must be expanded in terms of the fractional

deviations of the variables themselves. This is an immense algebraic task. The difficulty is that there is no assurance that this method of perturbation will work. The Liouville operator, $\mathcal{L}$, must be applied to each of the canonical variable deviations as well as to any new resultant monomials and other expressions produced. As the coefficients are identified, the basis set of vectors is chosen. The coefficients are taken as elements of the L matrix while noting any pattern that develops. The resultant L matrix must be studied for its natural partition structure to best determine the optimal truncation sizes. The natural truncation sizes chosen in this way are optimal, but no rigorous way of proving such an assertion has yet been found.

Assuming that reference values have been assigned to the Delaunay variables (oscillating Delaunay variables), the eigenvalues of the L matrix must be found and the structure of the eigenvalue spectrum ascertained. The Putzer algorithm must be applied to the smallest practical truncation of the L matrix and approximated results obtained. Data for successively higher orders of truncation should be obtained. All data should be compared to exact results as generated from a Runge-Kutta-Gill numerical integration procedure. Statistical error analysis for each truncation should be performed and a

phase space plot of corresponding coordinate-momentum deviations produced. This brief outline has given an indication of the amount of effort required.

The second area of additional work is the investigation of the extension of the Putzer technique to non-constant time dependent matrices. This could be accomplished by introducing a "forcing function" or inhomogeneous part to the defining differential equation for the scalar functions, r_K, used in the Putzer algorithm. This adds the capacity for handling time dependent Hamiltonians to the approximation scheme. The basic idea is to find functions of time, denoted by $s_j(t)$, which, when added to the defining differential equation system for the $r_j(t)$ scalar functions, i.e.

$$\dot{r}_j(t) = r_{j-1}(t) + \lambda_j r_j(t)$$

$$\text{with} \quad r_1(0) = 1 \quad \text{and} \quad r_j(0) = 0 \quad j \geq 2$$

gives the new system

$$\dot{r}_j(t) = r_{j-1}(t) + \lambda_j(t) r_j(t) + s_j(t)$$

$$\text{with} \quad r_1(0) = 1 \quad \text{and} \quad r_j(0) = 0 \quad j \geq 2 .$$

The $\lambda_j(t)$ are the time dependent eigenvalues of the time dependent matrix $A(t)$ and the $s_j(t)$ are unknown functions satisfying whatever initial conditions that are necessary. The new P matrices, denoted by P_j, also become functions of time if defined in analogous fashion with the original

Putzer definition. The new definition requires that the form

$$P_0(t=0) = \mathbf{I},$$

$$P_j(t) = \prod_{k=1}^{j} (A(t) - \lambda_k(t)\,\mathbf{I}) \qquad j = 1, 2, \ldots, n-1$$

be tried, where $\lambda_k(t)$ are the eigenvalues of A(t). It is not immediately clear that this approach will work since the time dependence is no longer separated out into distinct functions as in the original theorem. The new P matrices are now also time dependent. In the original theorem the P matrices carried information of where to apply the time dependence of the scalar functions, and while the new P matrices still play that role, they have been weighted by their time dependence.

The third area for further work involves the identification of the complete time dependence employed by the Putzer algorithm on a particular problem. The question is whether a function of time can be found that has the same series expansion of time produced by the usual application of the Putzer algorithm. The intent would be to determine if using the approximation scheme with the Putzer algorithm employing these new functions of time would produce results with reduced error. The multiplicities of the eigenvalues of the L matrix play a

crucial role. Recall from Chapter 2 that when an eigenvalue has an algebraic multiplicty of k, the $r_j(t)$ scalar functions of the Putzer algorithm have a time dependence of

$$r_j \sim \left[t^k \right] .$$

This behavior is the foremost contributing factor in explaining why the approximation method begins to fail at intermediate time ranges. Such behavior has been noted by Douglas in studying Lie algebraic methods applied to a one dimensional perturbed harmonic oscillator (Douglas, 1982:82). This property may force any practical utilization of the approximation scheme to be re-initialized using a new updated initial condition vector valid at a later time. When the eigenvalues have real parts, as in the case occurring in Chapter 4 when drag was introduced, the above behavior, while still present, is attenuated. The degree of attenuation depends upon the truncation size of the L matrix. By examining the sequence of coefficients belonging to the powers of t introduced into the scalar r_k functions by the presence of multiple eigenvalues of the L matrix, the hope is that the resulting series expansion can be identified as the expansion of some known function. This function, carrying all of the time dependence, would then be used in place of

the corresponding r_K function of the Putzer algorithm.

The fourth area constitutes the formulation of the details of the implementation of the approximation scheme to a practical, realistic problem. The first thing to be done is the determination of the optimal truncation size, N, of the L matrix. This truncation size will, in part, be determined by the accuracy level required. The next consideration concerns the multiplicities of the eigenvalues. The existence of multiple eigenvalues requires that the approximation method be re-started or reinitialized using a new updated coordinate vector for the initial condition vector. The specifics of when to do this depend upon the problem being solved and the accuracy desired. Instead of using numerical integration to determine the $r_J(t)$ functions, the N $r_J(t)$ functions should be found analytically, in addition to the first two rows of the N P_J matrices. Once the reinitialization time has been determined, a time step whose magnitude is one k^{th} of the reinitialization time can be chosen to step through the approximation scheme to the next reinitialization time in k steps. The advantage would be a computationally fast algorithm. For example, choosing the perturbed Kepler problem as the realistic practical problem, would allow the efficiency and accuracy of the approximation scheme to be

compared to the efficiency and accuracy of standard Cowell methods.

This dissertation has been a feasibility study performed numerically to judge how the Liouville operator can be represented by matrix techniques and applied to both simple and complicated problems from classical mechanics. Much has been learned. Much remains to be done. A deeper understanding of the mathematical mechanism by which solutions from larger and larger matrices converge to the exact solution needs to be gained. The effects of the different structures of the eigenvalue spectra encountered in the different problems need to be understood better. In many respects the research of this dissertation opens more questions than it answers. Yet the work presented here is a start. The work has shown methods that produce limited successful results. Very significant problems remain, but it is hoped that the research contained herein can lead to a practical methodology.

LITERATURE CITED

Bate, Roger R., Mueller, Donald D., and White, Jerry E., Fundamentals of Astrodynamics, Dover Publications, Inc., New York, 1971

Bedford, T. J., and Swift, J. W., Ed., New Directions in Dynamical Systems, Cambridge University Press, Cambridge, 1988

Bowen, Samuel P., "The abstract Hilbert space generalization of Feenberg's perturbation theory--A new method of quantum field theory", Journal of Mathematical Physics, Vol. 16, No. 3, 1975, pp. 620-632.

Bowen, Samuel P., Williams, Clayton D., and Mancini, Jay D. "Convergent Methods for calculating thermodynamic Green functions", Physical Review A, Vol. 30, No. 2, 1984, pp. 932-940.

Brouwer, Dirk, "Solution of the Problem of Artificial Satellite Theorey without Drag", The Astronomical Journal, Vol. 64, No. 1274, 1959, pp. 378-397.

Brouwer, Dirk and Clemence, Gerald M., Methods of Celestial Mechanics, Academic Press Inc., New York, 1961

Butcher, J. C., The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods, John Wiley & Sons, New York, New York, 1987, pp. 181-184.

Cayley, Authur, "Tables of the Developments of Functions in the Theory of Elliptic Motion", Mathematical Papers, Vol. 3, University Press, Cambridge, 1890, pp. 360-474.

Cody, William L. Jr., and Waite, William, Software Manual for the Elementary Functions, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1980, pp. 258-264.

Deutsch, Ralph, Orbital Dynamics of Space Vehicles, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1963

Dragt, Alex J., "Lectures on Nonlinear Orbit Dynamics", AIP Conference Proceedings, Vol. 87, American Institute of Physics, New York, 1982

Douglas, David R., "Lie Algebraic Methods for Particle Accelerator Theory", Ph.D. Dissertation, 1982, University of Maryland, 8308735 University Microfilms International, Ann Arbor, MI., 1985

Erugin, Nikolai P., Linear Systems of Ordinary Differential Equations with Periodic and Quasi-Periodic Coefficients, Translated by Scripta Technica, Inc. Richard Bellman, Translation Editor, Academic Press, New York London, 1966

Fitzpatrick, Philip M., Principles of Celestial Mechanics, Academic Press Inc., New York, New York, 1970

Geyling, Franz T. and Westerman, H. Robert, Introduction to Orbital Mechanics, Addison-Wesley Publishing Company, Reading, MA., 1971

Goldstein, Herbert, Classical Mechanics, Second Edition, Addison-Wesley Publishing Company, Reading, MA., 1980

Golub, Gene H., and Van Loan, Charles F., Matrix Computations, The Johns Hopkins University Press, Baltimore, Maryland, 1983

Kaula, William M., Theory of Satellite Geodesy, Blaisdell Publishing Company, Waltham, MA., 1966

Kovalevsky, Jean, Introduction to Celestial Mechanics, Springer-Verlag Inc., New York, New York., 1967

Masson, D., "Hilbert Space and the Padé Approximant", The Padé Approximant in Theoretical Physics, Baker, George A. Jr., and Gammel, John L., Eds., Academic Press, Inc., New York & London, 1970

Moler, Cleve, and Van Loan, Charles, "Nineteen Dubious Ways to Compute the Exponential of a Matrix", SIAM Review, Vol. 20, No. 4, 801-836, (1978)

Murphy, George M., Ordinary Differential Equations and Their Solutions, D. Van Nostrand Company, Inc., Princeton, New Jersey, 1960

Putzer, E. J., "Avoiding the Jordan Canonical Form in the Discussion of Linear Systems with Constant Coefficients", American Mathematical Monthly, Vol. 73 No. 1, 2-7, (1966)

Riesz, Frigyes, and Nagy, Béla Sz., Functional Analysis, Second Edition Translated by Leo F. Boron, Frederick Ungar Publishing Company, New York, 1955

Smith, B. T., Boyle, J. M., Dongarra, J. J., Garbow, B. S., Ikebe, Y., Klema, V. C., and Moler, C. B., Lecture Notes in Computer Science No. 6, Matrix Eigensystem Routines - EISPACK Guide, Second Edition, Springer-Verlag Inc., New York Berlin Heidelberg, 1976

Thompson, J. M. T., and Stewart H. B., Nonlinear Dynamics and Chaos; Geometrical Methods for Engineers and Scientists, John Wiley and Sons, New York, New York, 1986

Waltman, Paul, A Second Course in Elementary Differential Equations, Academic Press, New York, N. Y., 1985 pp. 49-61.

White, Frank M., Viscous Fluid Flow, McGraw-Hill Book Company, New York, 1974, p. 675-678.

APPENDIX A

FORTRAN LISTING FOR EIGENVALUE CALCULATION OF 56x56 L MATRIX FOR PERTURBED SHO

```
$DEBUG
      program eigval56
      parameter(nmax=56,npartshn=7)
      implicit real*8 (a-h,o-z)
      real*8 a(nmax,nmax), z(nmax,nmax), wr(nmax), wi(nmax), fv1(nmax)
      real*8 xm, xk, g
c     real zr(nmax,nmax), zi(nmax,nmax)
      real*8 aa(nmax,nmax)
      logical idoprint /.FALSE./
      logical lswap /.FALSE./
      logical dosort /.FALSE./
      integer n, nm, matz, iv1(nmax), ierr, kpartshn(npartshn)
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      character*24 filename
      character*24 filename1
      character*1  manswer
      character*1  idosort
c-->     Define partioning structure of the matrix...
c-->  Ending partition indices are given by kpartshn(j) for
c                                              j = 1,...,npartshn
c     Beginning partition indices are given by kpartshn(l)+1 for
c                                           l = 1,...,npartshn-1
      do 450 j = 1, npartshn
        kpartshn(j) = j*(j+1)
  450 continue
      write(*,'('' Input the values of m, k, g: (in SX.XXD+00) format'')
     &')
      write(*,'('' SX.XXD+00 SX.XXD+00 SX.XXD+00''')')
      read(*,1111) xm, xk, g
 1111 format(D9.4,1x,D9.4,1x,D9.4)
      write(*,'('' Input the dimension(order) .le.'',I3,'', of the A mat
     &rix.'')' ) kpartshn(npartshn)
      read(*,*) n
      write(*,'('' Do you want the matrix printed out? y or n '')')
      read(*,130) manswer
      if((manswer .eq. 'y' ) .or. (manswer .eq. 'Y')) idoprint = .TRUE.
      write(*,'('' Do you want the eigenvalues sorted in monotonic decr
     & order? (y or n) '')')
      read(*,130) idosort
      if((idosort .eq. 'y' ) .or. (idosort .eq. 'Y')) dosort = .TRUE.
c-->       now get filename to use in writing the output...
```

```
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(4,file=filename,status='new')
c-->        now get filename to write eigenvalues to be read by another
c             program.......
      write(*,'('' Please enter the filename to use to write data for pr
     -ogram accessability.'')' )
      read(*,160) filename1
      open(5,file=filename1,status='new')
c-->  CALL GETDAT AND GETTIM (OBTAIN SYSTEM DATE AND TIME)
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
      write(4,260)
      write(4,261) xm, xk, g
      if(dosort) write(4,270)
      write(4,265) filename
      write(4,266) imon, iday, iyr,  ihr, iminute, isecond
      write(5,260)
      write(5,261) xm, xk, g
      if(dosort) write(5,270)
      write(5,265) filename1
      write(5,266) imon, iday, iyr,  ihr, iminute, isecond
c         ....write data to output file for records ...
  130 format(A1)
  160 format(A24)
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 format(1H ,' XM IS:',D10.4,'  XK IS: ',D10.4,' G IS: ',D10.4)
  265 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  266 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
  270 format(1H ,' EIGENVALUES PROCESSED TO RUN FROM LARGEST TO SMALLEST
     - WITHIN EACH',/,'   SUB-PARTITION OF THE L MATRIX.')
c-->     Define the A matrix for 56x56 truncation......
      do 1000 j = 1, nmax
        do 1000 i = 1, nmax
          a(i,j) = 0.0D+00
 1000 continue
c-->    Define L (Upper Hessenberg) matrix for up to n=56 truncation....
      a(1,2) = 1.0D+00/xm
      a(2,1) = -xk

      a(2,3) = -g/6.0D+00
      a(3,4) = 3.0D+00/xm
      a(4,3) = -xk
      a(4,5) = 2.0D+00/xm
      a(5,4) = -(2.0D+00)*xk
      a(5,6) = 1.0D+00/xm
```

```
a(6,5) = (-3.0D+00)*xk

a(4,7) = -g/(6.0D+00)
a(5,8) = -g/3.0D+00
a(6,9) = -g/2.0D+00
a(7,8) =  (5.0D+00)/xm
a(8,7) = -xk
a(8,9) = (4.0D+00)/xm
a(9,8) = (-2.0D+00)*xk
a(9,10) = (3.0D+00)/xm
a(10,9) = (-3.0D+00)*xk
a(10,11) = (2.0D+00)/xm
a(11,10) = (-4.0D+00)*xk
a(11,12) = (1.0D+00)/xm
a(12,11) = (-5.0D+00)*xk

a(8,13) = -g/6.0D+00
a(9,14) = -g/3.0D+00
a(10,15) = -g/2.0D+00
a(11,16) = (-2.0D+00)*g/3.0D+00
a(12,17) = (-5.0D+00)*g/6.0D+00
a(13,14) = 7.0D+00/xm
a(14,13) = -xk
a(14,15) = 6.0D+00/xm
a(15,14) = (-2.0D+00)*xk
a(15,16) = 5.0D+00/xm
a(16,15) = (-3.0D+00)*xk
a(16,17) = 4.0D+00/xm
a(17,16) = (-4.0D+00)*xk
a(17,18) = 3.0D+00/xm
a(18,17) = (-5.0D+00)*xk
a(18,19) = 2.0D+00/xm
a(19,18) = (-6.0D+00)*xk
a(19,20) = 1.0D+00/xm
a(20,19) = (-7.0D+00)*xk
a(14,21) = -g/6.0D+00
a(15,22) = -g/3.0D+00
a(16,23) = -g/2.0D+00
a(17,24) = (-2.0D+00)*g/3.0D+00
a(18,25) = (-5.0D+00)*g/6.0D+00
a(19,26) = -g
a(20,27) = (-7.0D+00)*g/6.0D+00
a(21,22) = 9.0D+00/xm
a(22,21) = -xk
a(22,23) = (8.0D+00)/xm
a(23,22) = (-2.0D+00)*xk
a(23,24) = 7.0D+00/xm
a(24,23) = (-3.0D+00)*xk
```

```
a(24,25) = 6.0D+00/xm
a(25,24) = (-4.0D+00)*xk
a(25,26) = 5.0D+00/xm
a(26,25) = (-5.0D+00)*xk
a(26,27) = 4.0D+00/xm
a(27,26) = (-6.0D+00)*xk
a(27,28) = 3.0D+00/xm
a(28,27) = (-7.0D+00)*xk
a(28,29) =  2.0D+00/xm
a(29,28) = (-8.0D+00)*xk
a(29,30) =  1.0D+00/xm
a(30,29) = (-9.0D+00)*xk
a(22,31) = -g/6.0D+00
a(23,32) = -g/3.0D+00
a(24,33) = -g/2.0D+00
a(25,34) = (-2.0D+00)*g/3.0D+00
a(26,35) = (-5.0D+00)*g/6.0D+00
a(27,36) = -g
a(28,37) = (-7.0D+00)*g/6.0D+00
a(29,38) = (-4.0D+00)*g/3.0D+00
a(30,39) = (-3.0D+00)*g/2.0D+00
a(31,32) =  11.0D+00/xm
a(32,31) =  -xk
a(32,33) =  10.0D+00/xm
a(33,32) =  (-2.0D+00)*xk
a(33,34) =   9.0D+00/xm
a(34,33) =  (-3.0D+00)*xk
a(34,35) =   8.0D+00/xm
a(35,34) =  (-4.0D+00)*xk
a(35,36) =   7.0D+00/xm
a(36,35) =  (-5.0D+00)*xk
a(36,37) =   6.0D+00/xm
a(37,36) =  (-6.0D+00)*xk
a(37,38) =   5.0D+00/xm
a(38,37) =  (-7.0D+00)*xk
a(38,39) =   4.0D+00/xm
a(39,38) =  (-8.0D+00)*xk
a(39,40) =   3.0D+00/xm
a(40,39) =  (-9.0D+00)*xk
a(40,41) =   2.0D+00/xm
a(41,40) = (-10.0D+00)*xk
a(41,42) =   1.0D+00/xm
a(42,41) = (-11.0D+00)*xk
a(32,43) = -g/6.0D+00
a(33,44) = -g/3.0D+00
a(34,45) = -g/2.0D+00
a(35,46) = (-2.0D+00)*g/3.0D+00
a(36,47) = (-5.0D+00)*g/6.0D+00
```

```
      a(37,48) = -g
      a(38,49) = (-7.0D+00)*g/6.0D+00
      a(39,50) = (-4.0D+00)*g/3.0D+00
      a(40,51) = (-3.0D+00)*g/2.0D+00
      a(41,52) = (-5.0D+00)*g/3.0D+00
      a(42,53) = (-11.0D+00)*g/6.0D+00
      a(43,44) = 13.0D+00/xm
      a(44,43) = -xk
      a(44,45) = 12.0D+00/xm
      a(45,44) = (-2.0D+00)*xk
      a(45,46) = 11.0D+00/xm
      a(46,45) = (-3.0D+00)*xk
      a(46,47) = 10.0D+00/xm
      a(47,46) = (-4.0D+00)*xk
      a(47,48) = 9.0D+00/xm
      a(48,47) = (-5.0D+00)*xk
      a(48,49) = 8.0D+00/xm
      a(49,48) = (-6.0D+00)*xk
      a(49,50) = 7.0D+00/xm
      a(50,49) = (-7.0D+00)*xk
      a(50,51) = 6.0D+00/xm
      a(51,50) = (-8.0D+00)*xk
      a(51,52) = 5.0D+00/xm
      a(52,51) = (-9.0D+00)*xk
      a(52,53) = 4.0D+00/xm
      a(53,52) = (-10D+00)*xk
      a(53,54) = 3.0D+00/xm
      a(54,53) = (-11.0D+00)*xk
      a(54,55) = 2.0D+00/xm
      a(55,54) = (-12.0D+00)*xk
      a(55,56) = 1.0D+00/xm
      a(56,55) = (-13.0D+00)*xk
      if(idoprint) then
c-->     Write out A matrix....
        write(*,'('' Portion of A matrix used is:  '')' )
        write(4,'('' Portion of A matrix used is:  '')' )
        do 1180 k = 1, n
          write(*,'('' row '',i3)') k
          write(*,'('' '',8(f8.4,1x),/)' ) (a(k,j), j = 1,n)
 1180   continue
        do 1190 k = 1, n
          write(4,'('' row '',i3)') k
          write(4,'('' '',8(f8.4,1x),/)' ) (a(k,j), j = 1,n)
 1190   continue
      endif
        write(*,'('' '',71(1H*))' )
        write(4,'('' '',71(1H*))' )
        write(5,'('' '',71(1H*))' )
```

```
c     ****************************************************************
c       Proceed to get eigenvalues of the A matrix.
c         matx = 0 get eigenvalues only; matz = 1 get eigenvectors also
      matz = 0
      nm = nmax
c-->    Use the driver routine RUHESS to get eigenvalues.
      call ruhess(nm,n,a,wr,wi,0,z,iv1,fv1,ierr)
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c-->     Unpack eigenvector values from the NxN Z array, per
c          instructions and code from Section 2.3-6 of the Eispack Guide
c          pages 88-89
c     do 150 k = 1, n
c       if(wi(k) .ne. 0.0) go to 110
c       do 100 j = 1, n
c         zr(j,k) = z(j,k)
c         zi(j,k) = 0.0
c 100   continue
c       go to 150
c 110   if(wi(k) .lt. 0.0) go to 130
c       do 120 j = 1, n
c         zr(j,k) = z(j,k)
c         zi(j,k) = z(j,k+1)
c 120   continue
c       go to 150
c 130   do 140 j = 1, n
c         zr(j,k) = zr(j,k-1)
c         zi(j,k) = -zi(j,k-1)
c 140   continue
c 150 continue
      if(ierr .ne. 0) then
        if(ierr .lt. 0) then
          write(*,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(4,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(5,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(*,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(4,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(5,'('' Terminating program---IERR is: '',i3)' ) ierr
          stop 00001
        elseif(ierr .gt. 0) then
          write(*,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
```

```
          write(4,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(5,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(*,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(4,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(5,'('' Terminating program---IERR is: '',i3)' ) ierr
          stop 00002
        else
        endif
       else
c-->     IERR is zero, indicating no convergence problems--print results
        write(*,'('' '')' )
        write(4,'('' '')' )
c-->    Perform sort, if requested....
        if(dosort) then
c-->      Sort eigenvalue magnitudes on all partitions except the first
          do 2500 j = 1, npartshn - 1
            lswap = .FALSE.
            do 2800 k = kpartshn(j) + 1, kpartshn(j+1) - 3
            if( .not. (dabs(wr(k)) .gt. dabs(wr(k+2))  .or. dabs(wi(k))
     &        .gt. dabs(wi(k+2)) ) .and. .not. lswap ) then
              tempr = wr(k)
              wr(k) = wr(k+2)
              wr(k+1) = wr(k+3)
              wr(k+2) = tempr
              wr(k+3) = -tempr
              tempi = wi(k)
              wi(k) = wi(k+2)
              wi(k+1) = wi(k+3)
              wi(k+2) = tempi
              wi(k+3) = -tempi
              lswap = .TRUE.
            endif
2800        continue
2500      continue
        endif
        write(5,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
        do 300 i = 1, n
          if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &     (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) .or.
     &    (i.eq.(kpartshn(5)+1)) .or. (i.eq.(kpartshn(6)+1)) )
     &     write(*,'('' '')')
          if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &     (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) .or.
     &    (i.eq.(kpartshn(5)+1)) .or. (i.eq.(kpartshn(6)+1)) )
```

```
     &    write(4,'('' ')')
          write(*,'('' i = '',i3,'' EIGENVALUE: '',D16.10,1x,D16.10,''
     & (I) '')' ) i, wr(i), wi(i)
          write(4,'('' i = '',i3,'' EIGENVALUE: '',D16.10,1x,D16.10,''
     & (I) '')' ) i, wr(i), wi(i)
c-->          Write eigenvalues to output file for program use ....
          write(5,'(D16.10,2x,D16.10)') wr(i), wi(i)
  300   continue
        endif
        write(*,'('' '',71(1H*))' )
        write(4,'('' '',71(1H*))' )
        close(4)
        close(5)
        end
c******************** END of PROGRAM EIGVAL56 ************************
!t
$DEBUG
c************SUBROUTINE RUHESS (EISPACK Guide Modified RG) ***************
c************* Source listing taken from B.T. SMITH et. al. 1979 *********
c*************    Matrix Eigensystem Routines - EISPACK guide 2nd Ed *****
c*****************************************************************************
c
c     ******* MODIFIED 10/29/87 to only call balanc and hqr
c     *******  Since the input matrix is upper Hessenberg      TLH
c
      subroutine ruhess(nm,n,a,wr,wi,matz,z,iv1,fv1,ierr)
      implicit real*8 (a-h,o-z)
c
      integer n,nm,is1,is2,ierr,matz
      real*8 a(nm,n),wr(n),wi(n),z(nm,n),fv1(n)
      integer iv1(n)
c
      if(n .le. nm) go to 10
      ierr = 10 * n
      go to 50
c
   10 call balanc(nm,n,a,is1,is2,fv1)
c     call elmhes(nm,n,is1,is2,a,iv1)
      if(matz .ne. 0) go to 20
c     ***************** FIND EIGENVALUES ONLY ************************
      call hqr(nm,n,is1,is2,a,wr,wi,ierr)
      go to 50
c     ***************** FIND BOTH EIGENVALUES AND EIGENVECTORS ********
   20 call eltran(nm,n,is1,is2,a,iv1,z)
      call hqr2(nm,n,is1,is2,a,wr,wi,z,ierr)
      if(ierr .ne. 0) go to 50
      call balbak(nm,n,is1,is2,fv1,n,z)
   50 return
```

```
      end
c******************** END OF SUBROUTINE RUHESS ***************************
```

LISTINGS FOR SUBROUTINES BALANC, HQR, ELTRAN

AND BALBAK ARE CONTAINED IN

THE EISPACK GUIDE (SMITH et al, 1976)

APPENDIX B

FORTRAN LISTING FOR APPROXIMATION SCHEME EMPLOYING PUTZER ALGORITHM WITH 20x20 TRUNCATION OF THE L MATRIX FOR THE PERTURBED SHO

```
$DEBUG
      program putzer20
      parameter(nmax=20,npartshn=4)
      implicit real*8 (a-h,o-z)
      dimension a(nmax,nmax)
      dimension wr(nmax), wi(nmax)
      dimension linefo(6)
      complex*16 ca(nmax,nmax), ceigval(nmax), cident(nmax,nmax)
      complex*16 pfirst(nmax,nmax), psecond(nmax,nmax), temp(2,nmax)
      complex*16 expmatt(2,nmax)
      complex*16 u(nmax), u0(nmax), r_rkgill(nmax), r_last(nmax)
      integer n, nsteps, kpartshn(npartshn)
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      logical doprint /.FALSE./
      logical no_default_ev /.FALSE./
      logical ichgstep /.FALSE./
      logical initialze /.TRUE./
      character*24 filename
      character*24 ev_file
      character*12 linefo
      character*1  idoprint
      character*1  iread_ev_file
      character*1  istepans
      common /matrices/ ca, cident, n
      common /pmatrix/  pfirst, psecond
      data ev_file /'C:\eigval56.pgm        '/
      data h/.10D+00/
c-->     Define partioning structure of the matrix...
c-->  Ending partition indices are given by kpartshn(j) for
c                                              j = 1,...,npartshn
c     Beginning partition indices are given by kpartshn(l)+1 for
c                                           l = 1,...,npartshn-1
      do 450 j = 1, npartshn
        kpartshn(j) = j*(j+1)
  450 continue
      write(*,'('' Input the values of n, k, g:'',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YYbSd.dD+ZZ'' )')
      read(*,1229) XM, XK, G
 1229 format(1x,D8.4,1x,D8.4,1x,D8.4)
```

```
      write(*,'('' Input the dimension(order) .le.'',I3,'', of the A mat
     &rix.'')' ) kpartshn(npartshn)
      read(*,*) n
      write(*,'('' Input initial values XZERO, and PZERO: '',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YY'' )')
      read(*,1231) xzero, pzero
 1231 format(1x,D8.4,1x,D8.4)
c-->       now get filename to use in writing the output...
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(4,file=filename,status='new')
      write(*,'('' Do you wish to enter the path and filename of the fil
     &e containing the computed eigenvalues? (Y/N)'')')
      read(*,'(A1)') iread_ev_file
      if((iread_ev_file.eq.'Y').or.(iread_ev_file.eq.'y')) no_default_ev
     & = .TRUE.
      if(no_default_ev) then
        write(*,'('' Enter eigenvalue data PATH and FILENAME'')')
        read(*,'(A24)') ev_file
      endif
      open(7,file=ev_file,status='unknown')
      rewind 7
      write(*,'('' Input the init time, end time, and no.of steps.'',/,
     &'' Use the format:'',/,
     &'' D10.4,1X,D10.4,1X,I5'' ,/,
     &'' Sd.dddD+XXbSd.dddD+YYb54321'' )')
      read(*,1232) tbegin, tend, nsteps
 1232 format(1x,D10.4,1x,D10.4,1x,I5)
      write(*,'('' Default value of integration STEPSIZE is:'',D10.4)')
     & h
      write(*,'('' Do you wish to change it? Enter (Y/N)'')')
      read(*,'(A1)') istepans
      if((istepans .eq. 'y') .or. (istepans .eq. 'Y')) ichgstep = .TRUE.
      if(ichgstep) then
        write(*,'(1x,''Enter the value of integration STEPSIZE, use form
     &at'' )')
        write(*,'(1x,''Sd.ddddD+XX'' )')
        read(*,'(D11.4)') h
        write(*,'(D11.4)') h
      endif
      write(*,'('' Input the time at which to check calculations.'',/,
     &'' Use the format:'',/,
     &'' D11.4'' ,/,
     &'' Sdd.dddD+XX'' )')
      read(*,1233) tchkcalc
```

```
 1233 format(1x,D11.4)
      if(tchkcalc .gt. 0.0D+00) doprint = .TRUE.
c-->       Calculate the increment to advance the time with
      dt = (tend - tbegin)/dble(nsteps)
      t0 = tbegin
      t  = t0
c-->  CALL GETDAT AND GETTIM (OBTAIN SYSTEM DATE AND TIME)
c
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
      write(4,260)
      write(4,261) xm, xk, g
      write(4,262) xzero, pzero
      write(4,263)
      write(4,264) tbegin, tend, nsteps, dt, h
      write(4,270)
      write(4,265) filename
      write(4,266) imon, iday, iyr,  ihr, iminute, isecond
c         ....write data to output file for records ...
  160 format(A24)
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 format(1H ,' XM IS:',D10.4,'  XK IS: ',D10.4,' G IS: ',D10.4)
  262 format(1H ,' XZERO IS:',D10.4,'  PZERO IS: ',D10.4,' (USED IN BUIL
     &DING INITIAL CONDITION VECTOR)')
  263 format(1H ,' TRANSFORMATION BOUNDARIES FOLLOW: ')
  264 format(1H ,' TBEGIN = ',D11.6,' ; TEND = ',D11.6,' AND NSTEPS = ',
     -I6,/,5x,' DELTAT = ',D11.6,/,5x,' AND RKG STEPSIZE IS: ',D11.6)
  265 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  266 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
  270 format(1H ,' EIGENVALUES PROCESSED TO RUN FROM LARGEST TO SMALLEST
     - WITHIN EACH',/,'   PARTITION OF THE MATRIX.')
c-->     Define the A matrix for 20x20 truncation......
      do 1000 j = 1, n
        do 1010 i = 1, n
          a(i,j) = 0.0D+00
 1010   continue
 1000 continue
c-->    Define L (Upper Hessenberg) matrix for  n=20 truncation....
      a(1,2) = 1.0D+00/xm
      a(2,1) = -xk

      a(2,3) = -g/6.0D+00
      a(3,4) = 3.0D+00/xm
      a(4,3) = -xk
      a(4,5) = 2.0D+00/xm
      a(5,4) = -(2.0D+00)*xk
      a(5,6) = 1.0D+00/xm
```

```
      a(6,5) = (-3.0D+00)*xk

      a(4,7) = -g/(6.0D+00)
      a(5,8) = -g/3.0D+00
      a(6,9) = -g/2.0D+00
      a(7,8) =  (5.0D+00)/xm
      a(8,7) = -xk
      a(8,9) = (4.0D+00)/xm
      a(9,8) = (-2.0D+00)*xk
      a(9,10) = (3.0D+00)/xm
      a(10,9) = (-3.0D+00)*xk
      a(10,11) = (2.0D+00)/xm
      a(11,10) = (-4.0D+00)*xk
      a(11,12) = (1.0D+00)/xm
      a(12,11) = (-5.0D+00)*xk

      a(8,13) = -g/6.0D+00
      a(9,14) = -g/3.0D+00
      a(10,15) = -g/2.0D+00
      a(11,16) = (-2.0D+00)*g/3.0D+00
      a(12,17) = (-5.0D+00)*g/6.0D+00
      a(13,14) = 7.0D+00/xm
      a(14,13) = -xk
      a(14,15) = 6.0D+00/xm
      a(15,14) = (-2.0D+00)*xk
      a(15,16) = 5.0D+00/xm
      a(16,15) = (-3.0D+00)*xk
      a(16,17) = 4.0D+00/xm
      a(17,16) = (-4.0D+00)*xk
      a(17,18) = 3.0D+00/xm
      a(18,17) = (-5.0D+00)*xk
      a(18,19) = 2.0D+00/xm
      a(19,18) = (-6.0D+00)*xk
      a(19,20) = 1.0D+00/xm
      a(20,19) = (-7.0D+00)*xk

c     a(14,21) = -g/6.0D+00
c     a(15,22) = -g/3.0D+00
c     a(16,23) = -g/2.0D+00
c     a(17,24) = (-2.0D+00)*g/3.0D+00
c     a(18,25) = (-5.0D+00)*g/6.0D+00
c     a(19,26) = -g
c     a(20,27) = (-7.0D+00)*g/6.0D+00
c     a(21,22) = 9.0D+00/xm
c     a(22,21) = -xk
c     a(22,23) = (8.0D+00)/xm
c     a(23,22) = (-2.0D+00)*xk
c     a(23,24) = 7.0D+00/xm
```

```
c     a(24,23) = (-3.0D+00)*xk
c     a(24,25) = 6.0D+00/xn
c     a(25,24) = (-4.0D+00)*xk
c     a(25,26) = 5.0D+00/xn
c     a(26,25) = (-5.0D+00)*xk
c     a(26,27) = 4.0D+00/xn
c     a(27,26) = (-6.0D+00)*xk
c     a(27,28) = 3.0D+00/xn
c     a(28,27) = (-7.0D+00)*xk
c     a(28,29) =  2.0D+00/xn
c     a(29,28) = (-8.0D+00)*xk
c     a(29,30) =  1.0D+00/xn
c     a(30,29) = (-9.0D+00)*xk

c     a(22,31) = -g/6.0D+00
c     a(23,32) = -g/3.0D+00
c     a(24,33) = -g/2.0D+00
c     a(25,34) = (-2.0D+00)*g/3.0D+00
c     a(26,35) = (-5.0D+00)*g/6.0D+00
c     a(27,36) = -g
c     a(28,37) = (-7.0D+00)*g/6.0D+00
c     a(29,38) = (-4.0D+00)*g/3.0D+00
c     a(30,39) = (-3.0D+00)*g/2.0D+00
c     a(31,32) =  11.0D+00/xn
c     a(32,31) =  -xk
c     a(32,33) =  10.0D+00/xn
c     a(33,32) =  (-2.0D+00)*xk
c     a(33,34) =   9.0D+00/xn
c     a(34,33) =  (-3.0D+00)*xk
c     a(34,35) =   8.0D+00/xn
c     a(35,34) =  (-4.0D+00)*xk
c     a(35,36) =   7.0D+00/xn
c     a(36,35) =  (-5.0D+00)*xk
c     a(36,37) =   6.0D+00/xn
c     a(37,36) =  (-6.0D+00)*xk
c     a(37,38) =   5.0D+00/xn
c     a(38,37) =  (-7.0D+00)*xk
c     a(38,39) =   4.0D+00/xn
c     a(39,38) =  (-8.0D+00)*xk
c     a(39,40) =   3.0D+00/xn
c     a(40,39) =  (-9.0D+00)*xk
c     a(40,41) =   2.0D+00/xn
c     a(41,40) = (-10.0D+00)*xk
c     a(41,42) =   1.0D+00/xn
c     a(42,41) = (-11.0D+00)*xk

c     a(32,43) = -g/6.0D+00
c     a(33,44) = -g/3.0D+00
```

```
c     a(34,45) = -g/2.0D+00
c     a(35,46) = (-2.0D+00)*g/3.0D+00
c     a(36,47) = (-5.0D+00)*g/6.0D+00
c     a(37,48) = -g
c     a(38,49) = (-7.0D+00)*g/6.0D+00
c     a(39,50) = (-4.0D+00)*g/3.0D+00
c     a(40,51) = (-3.0D+00)*g/2.0D+00
c     a(41,52) = (-5.0D+00)*g/3.0D+00
c     a(42,53) = (-11.0D+00)*g/6.0D+00
c     a(43,44) = 13.0D+00/xm
c     a(44,43) = -xk
c     a(44,45) = 12.0D+00/xm
c     a(45,44) = (-2.0D+00)*xk
c     a(45,46) = 11.0D+00/xm
c     a(46,45) = (-3.0D+00)*xk
c     a(46,47) = 10.0D+00/xm
c     a(47,46) = (-4.0D+00)*xk
c     a(47,48) = 9.0D+00/xm
c     a(48,47) = (-5.0D+00)*xk
c     a(48,49) = 8.0D+00/xm
c     a(49,48) = (-6.0D+00)*xk
c     a(49,50) = 7.0D+00/xm
c     a(50,49) = (-7.0D+00)*xk
c     a(50,51) = 6.0D+00/xm
c     a(51,50) = (-8.0D+00)*xk
c     a(51,52) = 5.0D+00/xm
c     a(52,51) = (-9.0D+00)*xk
c     a(52,53) = 4.0D+00/xm
c     a(53,52) = (-10D+00)*xk
c     a(53,54) = 3.0D+00/xm
c     a(54,53) = (-11.0D+00)*xk
c     a(54,55) = 2.0D+00/xm
c     a(55,54) = (-12.0D+00)*xk
c     a(55,56) = 1.0D+00/xm
c     a(56,55) = (-13.0D+00)*xk

c-->     Write out A matrix....
      write(*,'('' Portion of A matrix used is:  '')' )
      write(4,'('' Portion of A matrix used is:  '')' )
      do 1180 k = 1, n
        write(*,'('' row '',i3)') k
        write(*,'('' '',6(D11.5,1x),/)' ) (a(k,j), j = 1,n)
 1180 continue
      do 1190 k = 1, n
        write(4,'('' row '',i3)') k
        write(4,'('' '',6(D11.5,1x),/)' ) (a(k,j), j = 1,n)
 1190 continue
      write(*,'('' '',71(1H*))' )
```

```
      write(4,'('' '',71(1H*))' )
c-->     convert matrix A into matrix CA, its complex form
      do 1230 jj = 1, n
        do 1240 ii = 1, n
          ca(ii,jj) = dcmplx( a(ii,jj), 0.0D+00 )
 1240   continue
 1230 continue
c-->     Form the vector of complex initial valued basis parameters
      u0(1) = dcmplx(xzero, 0.0D+00 )
      u0(2) = dcmplx(pzero, 0.0D+00 )

      u0(3) = dcmplx(xzero*xzero*xzero, 0.0D+00 )
      u0(4) = dcmplx(xzero*xzero*pzero, 0.0D+00 )
      u0(5) = dcmplx(xzero*pzero*pzero, 0.0D+00 )
      u0(6) = dcmplx(pzero*pzero*pzero, 0.0D+00 )

      u0(7) = dcmplx(xzero*xzero*xzero*xzero*xzero, 0.0D+00 )
      u0(8) = dcmplx(xzero*xzero*xzero*xzero*pzero, 0.0D+00 )
      u0(9) = dcmplx(xzero*xzero*xzero*pzero*pzero, 0.0D+00 )
      u0(10) = dcmplx(xzero*xzero*pzero*pzero*pzero, 0.0D+00 )
      u0(11) = dcmplx(xzero*pzero*pzero*pzero*pzero, 0.0D+00 )
      u0(12) = dcmplx(pzero*pzero*pzero*pzero*pzero, 0.0D+00 )

      u0(13) = dcmplx(xzero*xzero*xzero*xzero*xzero*xzero*xzero,0.0D+00)
      u0(14) = dcmplx(xzero*xzero*xzero*xzero*xzero*xzero*pzero,0.0D+00)
      u0(15) = dcmplx(xzero*xzero*xzero*xzero*xzero*pzero*pzero,0.0D+00)
      u0(16) = dcmplx(xzero*xzero*xzero*xzero*pzero*pzero*pzero,0.0D+00)
      u0(17) = dcmplx(xzero*xzero*xzero*pzero*pzero*pzero*pzero,0.0D+00)
      u0(18) = dcmplx(xzero*xzero*pzero*pzero*pzero*pzero*pzero,0.0D+00)
      u0(19) = dcmplx(xzero*pzero*pzero*pzero*pzero*pzero*pzero,0.0D+00)
      u0(20) = dcmplx(pzero*pzero*pzero*pzero*pzero*pzero*pzero,0.0D+00)
c-->     Build complex identity matrix of order n
      do 1250 jj = 1, n
        do 1260 ii = 1, n
          if(jj .eq. ii) then
            cident(ii,ii) = dcmplx(1.0D+00, 0.0D+00)
          else
            cident(ii,jj) = dcmplx(0.0D+00, 0.0D+00)
          endif
 1260   continue
 1250 continue
c--> ***************************************************************
c       Read in the eigenvalues from the file EIGVAL56.PGM.(default) or
c         from previously entered file.
c         First read past header containing info on physical constants
c         used to obtain the set of eigenvalues, and file creation info.
c
c-->      Read file until the header is passed.  The header is delimited
```

```
c-->      by a line that consists of the character string,
c-->      +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/
c-->      as a delimiter.
c-->      read to get past header on file known as 'ev_file'.
 1111 read(7,'(6(A12))',end=215) linefo
      if((linefo(2) .eq. '+/+/+/+/+/+/') .or. (linefo(2) .eq.
     $ '/+/+/+/+/+/+') ) then
        go to 500
      else
        go to 1111
      endif
c-->     re-set linefo.....
  500 do 335 ik = 1,6
        linefo(ik) = '            '
  335 continue
c--> we will open the file and read in the eigenvalues for the current
c-->      truncation of the L matrix.
      do 10 i = 1, n
        read(7,'(D16.10,2x,D16.10)',end=225) vr(i), vi(i)
   10   continue
c     ##### Form complex eigenvalue vector ......
        do 300 i = 1, n
          ceigval(i) = dcmplx( vr(i),vi(i) )
  300   continue
        if(doprint) then
          write(*,'('' Complex eigenvalue vector follows: '')')
          write(4,'('' Complex eigenvalue vector follows: '')')
          do 325 i = 1, n
            if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &        (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) )
     &        write(*,'(''    '')')
            if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &        (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) )
     &      write(4,'(''    '')')
            write(*,'(''  i = '',i3,'' EIGENVALUE: '',D14.8,1x,D14.8,''(
     &I)'')' ) i, vr(i), vi(i)
            write(4,'(''  i = '',i3,'' EIGENVALUE: '',D14.8,1x,D14.8,''(
     &I)'')' ) i, vr(i), vi(i)
  325     continue
        endif
c         Call routine BuildP to build two storage areas.  The first
c           PFIRST contains the first row of the P matrix having indice
c           INDEX, e.g. PFIRST(INDEX,row). The second, PSECOND, contains
c           the second row of the P matrix having indice INDEX, e.g.
c           PSECOND(INDEX,row).
      call buildp20(ceigval)
c-->      Write out the PFIRST & PSECOND matrices, if requested ......
      if(doprint) then
```

```
      write(*,'('' PFIRST & PSECOND Matrices follow:  '')' )
      write(4,'('' PFIRST & PSECOND Matrices follow:  '')' )
      do 3170 k = 1, n
        write(*,'('' P matrix index is '',i3)') k
          write(*,'('' row 1 '')')
          write(*,'('' '',6(G11.4,1x),/)' ) (pfirst(k,j), j = 1,n)
          write(*,'('' row 2 '')')
          write(*,'('' '',6(G11.4,1x),/)' ) (psecond(k,j), j = 1,n)
          write(*,'(1x,70(1H-))')
 3170 continue
      do 4170 k = 1, n
        write(4,'('' P matrix index is '',i3)') k
          write(4,'('' row 1 '')')
          write(4,'('' '',6(G11.4,1x),/)' ) (pfirst(k,j), j = 1,n)
          write(4,'('' row 2 '')')
          write(4,'('' '',6(G11.4,1x),/)' ) (psecond(k,j), j = 1,n)
          write(4,'(1x,70(1H-))')
 4170 continue
      write(*,'('' '',71(1H-))' )
      write(4,'('' '',71(1H-))' )
    endif
c     ***********  BEGIN ITERATING IN TIME **********************
    write(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
    write(4,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
    do 40 k=1,nsteps + 1
      tlast = t
      if((t .eq. 0.0D+00) .and. (initialze)) then
        r_rkgill(1) = dcmplx(1.0D+00, 0.0D+00)
        do 8200 kl = 2, n
          r_rkgill(kl) = dcmplx(0.0D+00, 0.0D+00)
 8200   continue
        initialze = .FALSE.
      else
c     Define last values of the r_rkgill(n) vector for use as the
c     initial condition vector for the RKGILL integration.
        do 8300 kn = 1, n
          r_last(kn) = r_rkgill(kn)
 8300   continue
c-->  Set the entire set of polynomial functions, each evaluated at the
c       integrated time T, given that the set of polynomials have the
c       initial conditions given by the values contained within the
c       the vector r_last(n).....
c
c     Pass DT to determine the number of times to execute the Runge-
c       Kutta-Gill integrator routine...
        call getrk20(h,n,ceigval,dt,r_last,r_rkgill,tlast,t)
c-----------------------------------------------------------------------
      endif
```

```
c       Initialize array to hold results of this particular time.
c          (iteration of deltat)
        do 3050 ii = 1, 2
          do 3060 jj = 1, n
            expmatt(ii,jj) = dcmplx(0.0D+00,0.0D+00)
 3060     continue
 3050   continue
        if((doprint) .and. (dabs(t - tchkcalc) .le. .02D+00)) then
          do 8400 kn = 1, n
            write(*,'('' For r sub '',i3,'', r is: '',D14.8,2x,D14.8)')
     &        kn, r_rkgill(kn)
 8400     continue
          do 8500 kn = 1, n
            write(4,'('' For r sub '',i3,'', r is: '',D14.8,2x,D14.8)')
     &        kn, r_rkgill(kn)
 8500     continue
        endif
c-->      Calculate the terms in the Putzer sum, accumulating the sum
c           continuously
        do 30000 index = 1, n

          do 3100 jj= 1, n
             temp(1,jj) =  r_rkgill(index)*pfirst(index,jj)
             temp(2,jj) =  r_rkgill(index)*psecond(index,jj)
 3100      continue
c         **** Calculate the term of the Putzer sum depending on index.
c         ****  Then add to the first index-1 sums stored in expmatt.
          do 3200 ii = 1, 2
            do 3210 jj = 1, n
              expmatt(ii,jj) = expmatt(ii,jj) + temp(ii,jj)
 3210       continue
 3200     continue
30000   continue
c-->  Initialize the u vector to zero.....
        do 50000 i = 1, n
          u(i) = dcmplx( 0.0D+00, 0.0D+00)
50000   continue
c-->   Form the coordinate vector for time t
        do 90100 i = 1, 2
          do 90110 j = 1, n
            u(i) = u(i) + expmatt(i,j)*u0(j)
90110     continue
90100   continue
        x    = dreal(u(1))
        pmom = dreal(u(2))
        write(4,'(1x,D11.4,3x,D17.9,2x,D17.9)' ) t, x, pmom
   40 continue
      write(*,'('' '',71(1H*))' )
```

```
      write(4,'('' '',71(1H*))' )
      close(4)
      go to 99999
  215 write(*,'('' EOF found in header of eigenvalue data file: '',A24)
     &') ev_file
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
      close(4)
      stop 00215
  225 write(*,'('' EOF found while reading eigenvalue data on file: '',
     &A24)') ev_file
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
      close(4)
      stop 00225
99999 continue
      end
c******************** END of PROGRAM PUTZER20 ************************
!t
$DEBUG
      subroutine buildp20(ceigval)
c*************************************************************************
c******* This subroutine builds the P matrix of the Putzer Algorithm. *
c******* Necessary eigenvalues are supplied via the calling line. The *
c******* range of complex matrix P is returned. The entire set of P ***
c******* matrices is calculated. ***************************************
c*************************************************************************
      parameter(nmax=20)
      implicit real*8 (a-h,o-z)
      integer n
      complex*16 p(nmax,nmax), f(nmax,nmax), pold(nmax,nmax)
      complex*16 pfirst(nmax,nmax), psecond(nmax,nmax)
      complex*16 ca(nmax,nmax), cident(nmax,nmax)
      complex*16 ceigval(nmax)
      common /matrices/ ca, cident, n
      common /pmatrix/  pfirst, psecond
c-->  Initialize the storage area o fthe first two rows of the entire
c       range of P matrices.  REMEMBER, index is the subscript of the
c       P matrix.
      do 110 jj = 1, n
        do 120 index = 1, n
           pfirst(index,jj) = dcmplx(0.0D+00, 0.0D+00)
           psecond(index,jj) = dcmplx(0.0D+00, 0.0D+00)
  120     continue
  110 continue
c-->    Define the first two rows of P sub 0 & store using index of 1
      pfirst(1,1) = dcmplx(1.0D+00, 0.0D+00)
      psecond(1,2) = dcmplx(1.0D+00, 0.0D+00)
```

```
c-->    Initialize the pold matrix to the complex identity matrix.
      do 400 jj = 1, n
        do 410 ii = 1, n
          pold(ii,jj) = cident(ii,jj)
  410   continue
  400 continue
c--->   Compute the remainder of the P matrices using recursion --------
      do 4000 index = 2, n
        do 500 jj = 1, n
          do 510 ii = 1, n
            if(ii.eq.jj) then
              f(ii,ii) = ca(ii,ii) - ceigval(index-1)
            else
              f(ii,jj) = ca(ii,jj)
            endif
  510     continue
  500   continue
c-->     Now compute P using the P matrix just calculated....
        do 600 ij = 1, n
          do 610 kj = 1, n
            do 700 kl = 1, n
              p(ij,kj) = p(ij,kj) + f(ij,kl)*pold(kl,kj)
  700       continue
  610     continue
  600   continue
c-->      Transfer P matrix to POLD matrix.....
        do 800 j = 1, n
          do 810 i = 1, n
            pold(i,j) = p(i,j)
  810     continue
  800   continue
c-->      Save first two rows of the P matrix sun INDEX for Putzer
c           calculations.......
        do 900 i = 1, n
          pfirst(index,i) = p(1,i)
          psecond(index,i) = p(2,i)
  900   continue
c-->      Re-initialize P matrix for matrix multiplication loops.
        do 1200 k = 1, n
          do 1210 j = 1, n
            p(j,k) = dcmplx(0.0D+00, 0.0D+00)
 1210     continue
 1200   continue
 4000 continue
      return
      end
c********************  END OF SUBROUTINE BUILDP20 **********************
!t
```

```
$DEBUG
      subroutine getrk20(h,n,cval,deltat,r_begin,r,begin_time,end_time)
c*************************************************************************
c*******  This subroutine finds the numerical values of the polynomial *
c*******  functions R sub K by performing Runge-Kutta integration upon *
c*******  the entire set of N simultaneous first-order linear **********
c*******  differential equations.  The integration is carried out from *
c*******  BEGIN_TIME to the value contained in the variable END_TIME. **
c*******  The variable R_BEGIN contains the values of R valid at the ***
c*******  time BEGIN_TIME, and thus contain the initial conditions of **
c*******  the integration.  ******************************************
c*******  The array CVAL contains the set of complex eigenvalues for ***
c*******  the truncation being solved.  The Runge-Kutta routine for N **
c*******  simultaneous linear first-order equations is taken from ******
c*******  Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c*******  1974 McGraw-Hill pp. 675-678 *********************************
c*******  The system to be solved can be represented by the following **
c*******  matrix equation. >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> ***
c*******                                                                **
c*******    -  -     -                                  -  -   -       **
c*******    |     |    | cval(1) 0    0 ..............0 | | r(1) |      **
c*******    |  _  |    |   1  cval(2) 0 ..............0 | | r(2) |      **
c*******    | dr  | =  |   0     1 cval(3) 0 ..........0 | | r(3) |      **
c*******    | --  |    |   0     0    1 cval(4) 0 ......0 | |   .  |      **
c*******    | dt  |    |   0     0    0   1 cval(5) 0...0 | |   .  |      **
c*******    |     |    |   .     .    .   0  \   \  0 ..0 | |   .  |      **
c*******    |     |    |   .     .    .   .    \   \ 0 .0 | |   .  |      **
c*******    |     |    |   0     0    0   0  ..  1 cval(n)| | r(n) |      **
c*******    -  -     -                                  -  -   -       **
c*******                                                                **
c******* ------------------------------------------------------------ ***
c*******  This subroutine builds only those polynomials for the 20x20 **
c*******  truncation of the L matrix. ********************************
c*************************************************************************
      parameter(nmax=20)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrol
      integer ul_steps, knt_steps
      complex*16  r(nmax), rdot(nmax), cval(nmax), r_begin(nmax)
      data t0/0.0D+00/
c-->  Initialize Runge routine variables, control index (m)
      m = 0
c-->  Determine the total number of steps (ul_steps) to make with Runge-
c       Kutta-Gill routine executing stepsize of H to advance solution
c         from begin_time to end_time.....
      yy = deltat/h
      if(yy .lt. 1.0D+00) stop 00111
      dtt = deltat
```

```
      klim = idint(yy)
      do 4545 jh = 1, klim + 2
        dtt = dtt - h
        if((dtt - 0.0D+00) .le. 1.0D-06) go to 4550
 4545 continue
 4550 ul_steps = jh
c--)  Initialize step counter (knt_steps).....
      knt_steps = 0
c--)    Initialize the r polynomial vector to values at time
c                          t = begin_time
      do 2300 jj = 1, n
        r(jj) = r_begin(jj)
 2300 continue
      t = begin_time
c     ----------------------------------------------------------------
    8 if(knt_steps - ul_steps) 6,7,7
    6 continue
      call runge20(n,r,rdot,t,h,m,kontrol)
      go to (10,20) kontrol
c       Define first-order derivatives of the system....
   10 rdot(1) = cval(1)*r(1)
      do 3300 kj = 2, n
        rdot(kj) = r(kj-1) + cval(kj)*r(kj)
 3300 continue
      go to 6
   20 continue
      knt_steps = knt_steps + 1
      go to 8
    7 continue
      end_time = t
      return
      end
c******************** END OF GETRK20 ********************************
!t
$DEBUG
      subroutine runge20(n,y,f,x,h,m,kontrl)
c*************************************************************************
c******** This subroutine performs Runge-Kutta integration by the Gills*
c******** method.  The Runge-Kutta routine for N ************************
c******** simultaneous linear first-order equations is taken from *******
c******** Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c******** 1974 McGraw-Hill pp. 675-678 ***********************************
c*************************************************************************
      parameter(nmax=20)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrl
      complex*16 y(nmax), f(nmax), q(nmax)
      n = n + 1
```

```
      go to (1,4,5,3,7) n
    1 do 2 i = 1, n
        q(i) = dcmplx(0.0D+00,0.0D+00)
    2 continue
      a = 0.5D+00
      go to 9
    3 a = 1.7071067811865475244D+00
    4 x = x + (.5D+00)*h
    5 do 6 i = 1, n
        y(i) = y(i) + a*(f(i)*h - q(i))
        q(i) = (2.0D+00)*a*h*f(i) + ((1.0D+00) - (3.0D+00)*a)*q(i)
    6 continue
      a = .29283218813452475 6D+00
      go to 9
    7 do 8 i = 1, n
        y(i) = y(i) + h*f(i)/(6.0D+00)  -  q(i)/(3.0D+00)
    8 continue
      n = 0
      kontrl = 2
      go to 10
    9 kontrl = 1
   10 continue
      return
      end
c****************  END OF SUBROUTINE RUNGE20  ************************
```

APPENDIX C

FORTRAN LISTING FOR "EXACT" RUNGE-KUTTA-GILL NUMERICAL INTEGRATION OF THE PERTURBED SHO

```
$DEBUG
      PROGRAM SOLVEHAM
C
C*************************************************************************
C*************************************************************************
C********                                                          ********
C********   This program solves a specified Hamiltonian by integrat- ********
C********     ing the resultant Hamilton's equations using a general-********
C********     ized fourth order RUNGE-KUTTA technique employing     ********
C********     GILL'S method. ***************************************
C********     The Runge-Kutta routine for N simultaneous linear     ********
C********     first-order equations is taken from Appendix C of the ********
C********     book VISCOUS FLUID FLOW by FRANK M. WHITE 1974 McGraw- ********
C********     Hill pp. 675-678 which implements GILL'S method of    ********
C********     Runge-Kutta integration. *********************************
C********   The program prompts the user for a filename (DOS) to    ********
C********      which it will write the data.  As written, at most   ********
C********      500 data points will be written.                     ********
C********                                                          ********
C********      H =  P*P/(2*M) + .5 K X*X + G*X*X*X*X/4!            ********
C********                                                          ********
C********                                                          ********
C*************************************************************************
C*************************************************************************
C
      PARAMETER(NMAX=2)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER*2 IHR,IMINUTE,ISECOND,I100TH,IYR,IMON,IDAY
      LOGICAL ICHGSTEP /.FALSE./
      LOGICAL INITIALZE /.TRUE./
      CHARACTER*24 FILENAME
      CHARACTER*1  ISTEPANS
      DIMENSION  Y(2), Y_LAST(2)
      DATA H/.10D+00/
C-->  CALL GETDAT AND GETTIM (OBTAIN SYSTEM DATE AND TIME)
      CALL GETDAT (IYR,IMON,IDAY)
      CALL GETTIM (IHR,IMINUTE,ISECOND,I100TH)
C          ....READ SYSTEM DATA
      WRITE(*,'('' Input the values of m, k, g:'',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YYbSd.dD+ZZ'' )')
```

```
      READ(*,1230) XM, XK, G
 1230 FORMAT(1X,D8.4,1X,D8.4,1X,D8.4)
      WRITE(*,'('' Input initial values XZERO, and PZERO: '',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YY'' )')
      READ(*,1231) XZERO, PZERO
 1231 FORMAT(1X,D8.4,1X,D8.4)
      WRITE(*,'('' Input the init time, end time, and no.of steps.'',/,
     &'' Use the format:'',/,
     &'' D10.4,1X,D10.4,1X,I5'' ,/,
     &'' Sd.dddD+XXbSd.dddD+YYb54321'' )')
      READ(*,1232) TBEGIN, TEND, NSTEPS
 1232 FORMAT(1X,D10.4,1X,D10.4,1X,I5)
      WRITE(*,'('' Default value of integration STEPSIZE is:'',D10.4)')
     & H
      WRITE(*,'('' Do you wish to change it? Enter (Y/N)'')')
      READ(*,'(A1)') ISTEPANS
      IF((ISTEPANS .EQ. 'y') .OR. (ISTEPANS .EQ. 'Y')) ICHGSTEP = .TRUE.
      IF(ICHGSTEP) THEN
        WRITE(*,'(1x,''Enter the value of integration STEPSIZE, use form
     &at'' )')
        WRITE(*,'(1x,''Sd.ddddD+XX'' )')
        READ(*,'(D11.4)') H
        WRITE(*,'(D11.4)') H
      ENDIF
C-->       CALCULATE THE INCREMENT TO ADVANCE THE TIME WITH
      DELTAT = (TEND - TBEGIN)/DBLE(NSTEPS)
      TO = TBEGIN
C-->  Initialize Runge routine variables, control index (M)
      M = 0
C-->        now get filename, starting time and time step for x and p
      WRITE(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      READ(*,'(A24)') FILENAME
C-->     ...OPEN FILE TO SAVE INTEGRATED VALUES OF Y(1), AND Y(2) (X,P)
      OPEN(6,FILE=FILENAME,STATUS='NEW')
C         ....write data to output file for records ...
  259 FORMAT(1H ,'RUNNING 4TH ORDER RUNGE-KUTTA GILL INTEGRATION ON THE
     $DATA BELOW.')
  260 FORMAT(1H ,'SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 FORMAT(1H ,'XM IS:',D10.4,'  XK IS: ',D10.4,' G IS: ',D10.4)
  262 FORMAT(1H ,'XZERO IS: ',D10.4,' ; PZERO IS: ',D10.4)
  263 FORMAT(1H ,'INTEGRATION LIMITS FOLLOW;        NSTEPS IS ',I6)
  264 FORMAT(1H ,'TBEGIN = ',D11.6,' ; TEND = ',D11.6,' DELTAT = ',
     - D11.6,/,5X,' AND RKG STEPSIZE IS: ',D11.6)
  265 FORMAT(1H ,'OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  266 FORMAT(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
```

```
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
C         ....WRITE DATA TO SCREEN FOR VERIFICATION ...
      WRITE(*,259)
      WRITE(*,260)
      WRITE(*,261) XM, XK, G
      WRITE(*,262) XZERO, PZERO
      WRITE(*,263) NSTEPS
      WRITE(*,264) TBEGIN, TEND, DELTAT, H
      WRITE(*,265) FILENAME
      WRITE(*,266) IMON, IDAY, IYR,  IHR, IMINUTE, ISECOND
      WRITE(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
C-->    WRITE DATA TO OUTPUT FILE...
      WRITE(6,259)
      WRITE(6,260)
      WRITE(6,261) XM, XK, G
      WRITE(6,262) XZERO, PZERO
      WRITE(6,263) NSTEPS
      WRITE(6,264) TBEGIN, TEND, DELTAT, H
      WRITE(6,265) FILENAME
      WRITE(6,266) IMON, IDAY, IYR,  IHR, IMINUTE, ISECOND
      WRITE(6,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
C
C          .....INITIALIZE NECESSARY VARIABLES
      N = NMAX
      T0 = 0.0
      T = T0
C-----------------------------------------------------------------------
      DO 44 K=1, NSTEPS + 1
        T_LAST = T
        IF((T .EQ. 0.0) .AND. (INITIALZE)) THEN
          Y(1) = XZERO
          Y(2) = PZERO
          INITIALZE = .FALSE.
        ELSE
C-->  Define last values of the Y(N) vector for use as the
C         initial condition vector for the RKGill integration.
          Y_LAST(1) = Y(1)
          Y_LAST(2) = Y(2)
C-->  For the time T, get the entire set of the Y(n)
C         functions, each evaluated at the time T, given that the set
C         has the initial conditions given by the
C         values contained within the vector Y_LAST(n).....
C
C     Pass DT to determine the number of times to execute Runge-Kutta-
C         Gill routine
          CALL DORKG(N,XM,XK,G,DELTAT,H,Y_LAST,Y,T_LAST,T)
C-----------------------------------------------------------------------
        ENDIF
```

```
C--> Write values to the output file.....
       WRITE(6,'(1x,D12.6,4x,D15.9,4x,D15.9)' ) T, Y(1), Y(2)
   44 CONTINUE
C       ....CLOSE OUTPUT FILE.....
      WRITE(*,'('' '',71(1H*))' )
      WRITE(6,'('' '',71(1H*))' )
      ENDFILE 6
      REWIND 6
      CLOSE(6)
      END
C************************** END OF SOLVEHAM ************************
$DEBUG
      SUBROUTINE DORKG(N,XM,XK,G,DT,H,Y_BEGIN,Y,T_BEGIN,T_END)
C*********************************************************************
C*******                                                        ***
C******* THIS ROUTINE SERVES AS THE INTERFACE TO THE RUNGE-KUTTA-GILL***
C*******  SUBROUTINE rkgill.  IT CONTROLS THE NUMBER OF TIMES THAT   ***
C*******  THE RUNGE STEPS OF rkgill ARE EXECUTED. IT IS DESIGNED TO  ***
C*******  ADVANCE THE SOLUTION OF Y AT TIME T_LAST, Y_LAST, TO THE   ***
C*******  THE SOLUTION VALID AT TIME T, GIVEN BY Y. ******************
C*********************************************************************
C*********************************************************************
      PARAMETER(NMAX=2)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER M, N, KONTROL
      INTEGER DELTAT_STEPS, KNT_STEPS
      DIMENSION F(2), Y(2), Y_BEGIN(2)
      DATA FACT3/6.0D+0/, FACT4/24.0D+0/
C--> Determine the total number of steps (DELTAT_STEPS) to make with
C       Runge-Kutta-Gill routine executing stepsize of H to advance
C       solution by amount DELTAT, from T_BEGIN to T.....
      YY = DT/H
      IF(YY .LT. 1.0) STOP 00111
      DTT = DT
      KLIM = IDINT(YY)
C       Do loop used to get precise correct value for DELTAT_STEPS
      DO 4545 JH = 1, KLIM + 2
        DTT = DTT - H
        IF((DTT - 0.0D+00) .LE. 1.0D-06) GO TO 4550
 4545 CONTINUE
 4550 DELTAT_STEPS = JH
C--> Initialize Runge routine variables, control index (m)
      M = 0
C--> Initialize step counter (knt_steps).....
      KNT_STEPS = 0
      T = T_BEGIN
      Y(1) = Y_BEGIN(1)
      Y(2) = Y_BEGIN(2)
```

```
C-----------------------------------------------------------------
C
C       ....CALL ON 4-TH ORDER RUNGE-KUTTA-GILLS FUNCTION
    8 IF(KNT_STEPS - DELTAT_STEPS) 6,7,7
    6 CONTINUE
      CALL RKGILL(N,Y,F,T,H,M,KONTROL)
      GO TO (10,20) KONTROL
c       Define first-order derivatives of the system....
   10 F(1) = Y(2)/XM
      F(2) = -(XK + (G*Y(1)*Y(1))/FACT3)*Y(1)
      GO TO 6
   20 CONTINUE
      KNT_STEPS = KNT_STEPS + 1
      GO TO 8
    7 CONTINUE
      T_END = T
      RETURN
      END
C********************  END OF DORKG ***********************************
$DEBUG
      subroutine rkgill(n,y,f,x,h,m,kontrl)
c***************************************************************************
c*******  This subroutine performs Runge-Kutta integration by the Gills*
c*******  method.  The Runge-Kutta routine for N ************************
c*******  simultaneous linear first-order equations is taken from ******
c*******  Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c*******  1974 McGraw-Hill pp. 675-678 ********************************
c***************************************************************************
      parameter(nmax=2)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrl
      dimension  y(nmax), f(nmax), q(nmax)
      m = m + 1
      go to (1,4,5,3,7) m
    1 do 2 i = 1, n
        q(i) = 0.0D+00
    2 continue
      a = 0.5D+00
      go to 9
    3 a = 1.7071067811865475244D+00
    4 x = x + (.5D+00)*h
    5 do 6 i = 1, n
        y(i) = y(i) + a*(f(i)*h - q(i))
        q(i) = (2.0D+00)*a*h*f(i) + ((1.0D+00) - (3.0D+00)*a)*q(i)
    6 continue
      a = .29283218813452475 6D+00
      go to 9
    7 do 8 i = 1, n
```

```
        y(i) = y(i) + h*f(i)/(6.0D+00)  -  q(i)/(3.0D+00)
    8 continue
      m = 0
      kontrl = 2
      go to 10
    9 kontrl = 1
   10 continue
      return
      end
c****************  END OF SUBROUTINE RK6ILL  ***************************
```

APPENDIX D

FORTRAN LISTING FOR EIGENVALUE CALCULATION USING EISPACK HQR SUBROUTINES FOR 54x54 L MATRIX TRUNCATION OF PERTURBED SHO WITH DRAG FORCE

```
$DEBUG
      program ev54drag
      parameter(nmax=54,npartshn=9)
      implicit real*8 (a-h,o-z)
      real*8 a(nmax,nmax), z(nmax,nmax), wr(nmax), wi(nmax), fv1(nmax)
      real*8 xm, xk, g
c     real zr(nmax,nmax), zi(nmax,nmax)
      logical idoprint /.FALSE./
      logical lswap /.FALSE./
      logical dosort /.FALSE./
      integer n, nm, matz, iv1(nmax), ierr, kpartshn(npartshn)
      integer index_zero
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      character*24 filename
      character*24 filename1
      character*1  manswer
      character*1  idosort
      data fact3/6.0D+00/
c-->    Define partioning structure of the matrix...
c-->  Ending partition indices are given by kpartshn(j) for
c                                                  j = 1,...,npartshn
c     Beginning partition indices are given by kpartshn(l)+1 for
c                                             l = 1,...,npartshn-1
      do 450 j = 1, npartshn
        kpartshn(j) = ( (((j+1)*(j+2))/2) - 1)
  450 continue
      write(*,'('' Input the values of m, k, g: (in SX.XXD+00) format'')
     &')
      write(*,'('' SX.XXD+00 SX.XXD+00 SX.XXD+00'')')
      read(*,1111) xm, xk, g
 1111 format(D9.4,1x,D9.4,1x,D9.4)
      write(*,'('' Enter the value of alpha, the drag coefficient: (in S
     &X.XXXXXD+00) format'')')
      write(*,'('' SX.XXXXXD+00'')')
      read(*,1112) alpha
 1112 format(D12.5)
      write(*,'('' Input the dimension(order) .le.'',I3,'', of the A mat
     &rix.'')' ) kpartshn(npartshn)
      read(*,*) n
      write(*,'(''  Do you want the matrix printed out? y or n '')')
      read(*,130) manswer
```

```
      if((manswer .eq. 'y' ) .or. (manswer .eq. 'Y')) idoprint = .TRUE.
      write(*,'('' Do you want the eigenvalues sorted in monotonic decr
     & order? (y or n) '')')
      read(*,130) idosort
      if((idosort .eq. 'y' ) .or. (idosort .eq. 'Y')) dosort = .TRUE.
c-->        now get filename to use in writing the output...
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(4,file=filename,status='new')
c-->        now get filename to write eigenvalues to be read by another
c             program.......
      write(*,'('' Please enter the filename to use to write data for pr
     -ogram accessability.'')' )
      read(*,160) filename1
      open(5,file=filename1,status='new')
c-->  CALL GETDAT AND GETTIM (OBTAIN SYSTEM DATE AND TIME)
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
      write(4,180)
      write(4,260)
      write(4,261) xm, xk, g
      write(4,262) alpha
      if(dosort) write(4,270)
      write(4,265) filename
      write(4,266) imon, iday, iyr,  ihr, iminute, isecond
      write(5,180)
      write(5,260)
      write(5,261) xm, xk, g
      write(5,262) alpha
      if(dosort) write(5,270)
      write(5,265) filename1
      write(5,266) imon, iday, iyr,  ihr, iminute, isecond
c         ....write data to output file for records ...
  130 format(A1)
  160 format(A24)
  180 format(1H ,' OBTAINING EIGENVALUES OF MATRIX L WITH DRAG TERMS ADD
     &DED:')
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 format(1H ,' XM IS:',D10.4,'  XK IS: ',D10.4,' G IS: ',D10.4)
  262 format(1H ,' DRAG COEFFICIENT ALPHA IS: ',D12.5)
  265 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  266 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     -',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
  270 format(1H ,' EIGENVALUES PROCESSED TO RUN FROM LARGEST TO SMALLEST
     - WITHIN EACH',/,'   SUB-PARTITION OF THE L MATRIX.')
c-->     Define the A matrix for 54x54 truncation....Drag terms added..
      do 1000 j = 1, nmax
```

```
      do 1000 i = 1, nmax
        a(i,j) = 0.00D+00
 1000 continue
c-->   Define L (Upper Hessenberg) matrix for up to n=54 truncation....
      aom = alpha/xm
      a(1,2) = 1.00D+00/xm
      a(2,1) = -xk
      a(2,2) = -aom

      a(3,4) = 2.00D+00/xm
      a(4,3) = -xk
      a(4,4) = -aom
      a(4,5) = 1.00D+00/xm
      a(5,4) = -2.00D+00*xk
      a(5,5) = -2.00D+00*aom

      a(2,6) = -g/fact3
      a(6,7) =  3.00D+00/xm
      a(7,6) = -xk
      a(7,7) = -aom
      a(7,8) =  2.00D+00/xm
      a(8,7) =  -2.00D+00*xk
      a(8,8) =  -2.00D+00*aom
      a(8,9) = 1.00D+00/xm
      a(9,8) = -3.00D+00*xk
      a(9,9) = -3.00D+00*aom

      a(4,10) = -g/fact3
      a(5,11) = -2.00D+00*g/fact3
      a(10,11) = 4.00D+00/xm
      a(11,10) = -xk
      a(11,11) = -aom
      a(11,12) =  3.00D+00/xm
      a(12,11) = -2.00D+00*xk
      a(12,12) = -2.00D+00*aom
      a(12,13) =  2.00D+00/xm
      a(13,12) = -3.00D+00*xk
      a(13,13) = -3.00D+00*aom
      a(13,14) =  1.00D+00/xm
      a(14,13) = -4.00D+00*xk
      a(14,14) = -4.00D+00*aom

      a(7,15) =  -g/fact3
      a(8,16) =  -2.00D+00*g/fact3
      a(9,17) =  -3.00D+00*g/fact3
      a(15,16) =  5.00D+00/xm
      a(16,15) = -xk
      a(16,16) =  -aom
```

```
a(16,17) =  4.00D+00/xm
a(17,16) = -2.00D+00*xk
a(17,17) = -2.00D+00*aom
a(17,18) =  3.00D+00/xm
a(18,17) = -3.00D+00*xk
a(18,18) = -3.00D+00*aom
a(18,19) =  2.00D+00/xm
a(19,18) = -4.00D+00*xk
a(19,19) = -4.00D+00*aom
a(19,20) =  1.00D+00/xm
a(20,19) = -5.00D+00*xk
a(20,20) = -5.00D+00*aom

a(11,21) = -g/fact3
a(12,22) = -2.00D+00*g/fact3
a(13,23) = -3.00D+00*g/fact3
a(14,24) = -4.00D+00*g/fact3
a(21,22) =  6.00D+00/xm
a(22,21) = -xk
a(22,22) = -aom
a(22,23) =  5.00D+00/xm
a(23,22) = -2.00D+00*xk
a(23,23) = -2.00D+00*aom
a(23,24) =  4.00D+00/xm
a(24,23) = -3.00D+00*xk
a(24,24) = -3.00D+00*aom
a(24,25) =  3.00D+00/xm
a(25,24) = -4.00D+00*xk
a(25,25) = -4.00D+00*aom
a(25,26) =  2.00D+00/xm
a(26,25) = -5.00D+00*xk
a(26,26) = -5.00D+00*aom
a(26,27) =  1.00D+00/xm
a(27,26) = -6.00D+00*xk
a(27,27) = -6.00D+00*aom

a(16,28) = -g/fact3
a(17,29) = -2.00D+00*g/fact3
a(18,30) = -3.00D+00*g/fact3
a(19,31) = -4.00D+00*g/fact3
a(20,32) = -5.00D+00*g/fact3
a(28,29) =  7.00D+00/xm
a(29,28) = -xk
a(29,29) = -aom
a(29,30) =  6.00D+00/xm
a(30,29) = -2.00D+00*xk
a(30,30) = -2.00D+00*aom
a(30,31) =  5.00D+00/xm
```

```
a(31,30) = -3.00D+00*xk
a(31,31) = -3.00D+00*aom
a(31,32) =  4.00D+00/xm
a(32,31) = -4.00D+00*xk
a(32,32) = -4.00D+00*aom
a(32,33) =  3.00D+00/xm
a(33,32) = -5.00D+00*xk
a(33,33) = -5.00D+00*aom
a(33,34) =  2.00D+00/xm
a(34,33) = -6.00D+00*xk
a(34,34) = -6.00D+00*aom
a(34,35) =  1.00D+00/xm
a(35,34) = -7.00D+00*xk
a(35,35) = -7.00D+00*aom

a(22,36) = -g/fact3
a(23,37) = -2.00D+00*g/fact3
a(24,38) = -3.00D+00*g/fact3
a(25,39) = -4.00D+00*g/fact3
a(26,40) = -5.00D+00*g/fact3
a(27,41) = -6.00D+00*g/fact3
a(36,37) =  8.00D+00/xm
a(37,36) = -xk
a(37,37) = -aom
a(37,38) =  7.00D+00/xm
a(38,37) = -2.00D+00*xk
a(38,38) = -2.00D+00*aom
a(38,39) =  6.00D+00/xm
a(39,38) = -3.00D+00*xk
a(39,39) = -3.00D+00*aom
a(39,40) =  5.00D+00/xm
a(40,39) = -4.00D+00*xk
a(40,40) = -4.00D+00*aom
a(40,41) =  4.00D+00/xm
a(41,40) = -5.00D+00*xk
a(41,41) = -5.00D+00*aom
a(41,42) =  3.00D+00/xm
a(42,41) = -6.00D+00*xk
a(42,42) = -6.00D+00*aom
a(42,43) =  2.00D+00/xm
a(43,42) = -7.00D+00*xk
a(43,43) = -7.00D+00*aom
a(43,44) =  1.00D+00/xm
a(44,43) = -8.00D+00*xk
a(44,44) = -8.00D+00*aom

a(29,45) = -g/fact3
a(30,46) = -2.00D+00*g/fact3
```

```
      a(31,47) = -3.00D+00*g/fact3
      a(32,48) = -4.00D+00*g/fact3
      a(33,49) = -5.00D+00*g/fact3
      a(34,50) = -6.00D+00*g/fact3
      a(35,51) = -7.00D+00*g/fact3
      a(45,46) =  9.00D+00/xm
      a(46,45) = -xk
      a(46,46) = -aom
      a(46,47) =  8.00D+00/xm
      a(47,46) = -2.00D+00*xk
      a(47,47) = -2.00D+00*aom
      a(47,48) =  7.00D+00/xm
      a(48,47) = -3.00D+00*xk
      a(48,48) = -3.00D+00*aom
      a(48,49) =  6.00D+00/xm
      a(49,48) = -4.00D+00*xk
      a(49,49) = -4.00D+00*aom
      a(49,50) =  5.00D+00/xm
      a(50,49) = -5.00D+00*xk
      a(50,50) = -5.00D+00*aom
      a(50,51) =  4.00D+00/xm
      a(51,50) = -6.00D+00*xk
      a(51,51) = -6.00D+00*aom
      a(51,52) =  3.00D+00/xm
      a(52,51) = -7.00D+00*xk
      a(52,52) = -7.00D+00*aom
      a(52,53) =  2.00D+00/xm
      a(53,52) = -8.00D+00*xk
      a(53,53) = -8.00D+00*aom
      a(53,54) =  1.00D+00/xm
      a(54,53) = -9.00D+00*xk
      a(54,54) = -9.00D+00*aom

c     a(37,55) = -g/fact3
c     a(38,56) = -2.00D+00*g/fact3
c     a(39,57) = -3.00D+00*g/fact3
c     a(40,58) = -4.00D+00*g/fact3
c     a(41,59) = -5.00D+00*g/fact3
c     a(42,60) = -6.00D+00*g/fact3
c     a(43,61) = -7.00D+00*g/fact3
c     a(44,62) = -8.00D+00*g/fact3
c     a(55,56) =  10.00D+00/xm
c     a(56,55) = -xk
c     a(56,56) = -aom
c     a(56,57) =  9.00D+00/xm
c     a(57,56) = -2.00D+00*xk
c     a(57,57) = -2.00D+00*aom
c     a(57,58) =  8.00D+00/xm
```

```
c     a(58,57) = -3.00D+00*xk
c     a(58,58) = -3.00D+00*aom
c     a(58,59) =  7.00D+00/xm
c     a(59,58) = -4.00D+00*xk
c     a(59,59) = -4.00D+00*aom
c     a(59,60) =  6.00D+00/xm
c     a(60,59) = -5.00D+00*xk
c     a(60,60) = -5.00D+00*aom
c     a(60,61) =  5.00D+00/xm
c     a(61,60) = -6.00D+00*xk
c     a(61,61) = -6.00D+00*aom
c     a(61,62) =  4.00D+00/xm
c     a(62,61) = -7.00D+00*xk
c     a(62,62) = -7.00D+00*aom
c     a(62,63) =  3.00D+00/xm
c     a(63,62) = -8.00D+00*xk
c     a(63,63) = -8.00D+00*aom
c     a(63,64) =  2.00D+00/xm
c     a(64,63) = -9.00D+00*xk
c     a(64,64) = -9.00D+00*aom
c     a(64,65) =  1.00D+00/xm
c     a(65,64) = -10.00D+00*xk
c     a(65,65) = -10.00D+00*aom
      if(idoprint) then
c-->     Write out A matrix....
        write(*,'('' Portion of A matrix used is:  '')' )
        write(4,'('' Portion of A matrix used is:  '')' )
        do 1180 k = 1, n
          write(*,'('' row '',i3)') k
          write(*,'('' '',8(f8.4,1x),/)' ) (a(k,j), j = 1,n)
 1180   continue
        do 1190 k = 1, n
          write(4,'('' row '',i3)') k
          write(4,'('' '',8(f8.4,1x),/)' ) (a(k,j), j = 1,n)
 1190   continue
      endif
        write(*,'('' '',71(1H*))' )
        write(4,'('' '',71(1H*))' )
        write(5,'('' '',71(1H*))' )
c     ***********************************************************************
c       Proceed to get eigenvalues of the A matrix.
c         matx = 0 get eigenvalues only; matz = 1 get eigenvectors also
      matz = 0
      nm = nmax
c-->   Use the driver routine RUHESS to get eigenvalues.
      call ruhess(nm,n,a,wr,wi,0,z,iv1,fv1,ierr)
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c-->     Unpack eigenvector values from the NxN Z array, per
```

```
c         instructions and code from Section 2.3-6 of the Eispack Guide
c         pages 88-89
c     do 150 k = 1, n
c       if(wi(k) .ne. 0.0) go to 110
c       do 100 j = 1, n
c         zr(j,k) = z(j,k)
c         zi(j,k) = 0.0
c 100   continue
c       go to 150
c 110   if(wi(k) .lt. 0.0) go to 130
c       do 120 j = 1, n
c         zr(j,k) = z(j,k)
c         zi(j,k) = z(j,k+1)
c 120   continue
c       go to 150
c 130   do 140 j = 1, n
c         zr(j,k) = zr(j,k-1)
c         zi(j,k) = -zi(j,k-1)
c 140   continue
c 150 continue
      if(ierr .ne. 0) then
        if(ierr .lt. 0) then
          write(*,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(4,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(5,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(*,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(4,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(5,'('' Terminating program---IERR is: '',i3)' ) ierr
          stop 00001
        elseif(ierr .gt. 0) then
          write(*,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(4,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(5,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(*,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(4,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(5,'('' Terminating program---IERR is: '',i3)' ) ierr
```

```
          stop 00002
        else
        endif
      else
c-->     IERR is zero, indicating no convergence problems--print results
        write(*,'('' '')' )
        write(4,'('' '')' )
c-->    Perform sort, if requested....
        if(dosort) then
c-->      Sort eigenvalue magnitudes on all partitions except the first
c           three...
          do 2500 j = 4, npartshn
c             Check if number of elements in the partition is even or odd
            numelem = kpartshn(j) - kpartshn(j-1)
            if( mod(numelem,2) .eq. 0 ) then
c               EVEN number
              lswap = .FALSE.
              do 6800 k = kpartshn(j-1) + 1, kpartshn(j) - 3, 2
                if( .not. ( dabs(wi(k)) .gt. dabs(wi(k+2)) )
     &                                  .and. .not. lswap ) then
                  tempr = wr(k)
                  temprr = wr(k+1)
                  wr(k) = wr(k+2)
                  wr(k+1) = wr(k+3)
                  wr(k+2) = tempr
                  wr(k+3) = temprr
                  tempi = wi(k)
                  tempii = wi(k+1)
                  wi(k) = wi(k+2)
                  wi(k+1) = wi(k+3)
                  wi(k+2) = tempi
                  wi(k+3) = tempii
                  lswap = .TRUE.
                endif
 6800         continue
            else
c               ODD number in partition, one element has zero imag part
                index_zero = 0
                do 7800 k = kpartshn(j-1) + 1, kpartshn(j)
                  if( dabs(wi(k) - 0.00D+00) .le. 1.00D-12) index_zero=k
 7800           continue
c               Place the zero imag element at the end of the partition
                tempr = wr(kpartshn(j))
                tempi = wi(kpartshn(j))
                wr(kpartshn(j)) = wr(index_zero)
                wi(kpartshn(j)) = wi(index_zero)
                wr(kpartshn(j) - 2 ) = tempr
                wi(kpartshn(j) - 2 ) = -tempi
```

```
                wr(kpartshn(j) - 1 ) = tempr
                vi(kpartshn(j) - 1 ) = tempi
            endif
 2500     continue
        endif
        write(5,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
        do 300 i = 1, n
          if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &     (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) .or.
     &    (i.eq.(kpartshn(5)+1)) .or. (i.eq.(kpartshn(6)+1))  .or.
     &    (i.eq.(kpartshn(7)+1)) .or. (i.eq.(kpartshn(8)+1))  .or.
     &    (i.eq.(kpartshn(9)+1)) )
     &     write(*,'(''     '')')
          if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &     (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) .or.
     &    (i.eq.(kpartshn(5)+1)) .or. (i.eq.(kpartshn(6)+1))  .or.
     &    (i.eq.(kpartshn(7)+1)) .or. (i.eq.(kpartshn(8)+1))  .or.
     &    (i.eq.(kpartshn(9)+1)) )
     &     write(4,'(''     '')')
          write(*,'(''  i = '',i3,''  EIGENVALUE: '',D16.10,1x,D16.10,''
     & (I) '')' ) i, wr(i), vi(i)
          write(4,'(''  i = '',i3,''  EIGENVALUE: '',D16.10,1x,D16.10,''
     & (I) '')' ) i, wr(i), vi(i)
c-->          Write eigenvalues to output file for program use ....
          write(5,'(D16.10,2x,D16.10)') wr(i), vi(i)
  300   continue
      endif
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
      close(4)
      close(5)
      end
c*******************  END of PROGRAM EV54DRAG  *************************
:t
$DEBUG
c***********SUBROUTINE RUHESS (EISPACK Guide Modified RG) **************
c************ Source listing taken from B.T. SMITH et. al. 1979 ********
c************    Matrix Eigensystem Routines - EISPACK guide 2nd Ed ****
c***********************************************************************
c
c     *******  MODIFIED 10/29/87 to only call balanc and hqr
c     *******   Since the input matrix is upper Hessenberg      TLH
c
      subroutine ruhess(nm,n,a,wr,wi,matz,z,iv1,fv1,ierr)
      implicit real*8 (a-h,o-z)
      integer n,nm,is1,is2,ierr,matz
      real*8 a(nm,n),wr(n),wi(n),z(nm,n),fv1(n)
      integer iv1(n)
```

```
      if(n .le. nm) go to 10
      ierr = 10 * n
      go to 50
   10 call balanc(nm,n,a,is1,is2,fv1)
c     call elmhes(nm,n,is1,is2,a,iv1)
      if(matz .ne. 0) go to 20
c     ***************** FIND EIGENVALUES ONLY ************************
      call hqr(nm,n,is1,is2,a,wr,wi,ierr)
      go to 50
c     ***************** FIND BOTH EIGENVALUES AND EIGENVECTORS ********
   20 call eltran(nm,n,is1,is2,a,iv1,z)
      call hqr2(nm,n,is1,is2,a,wr,wi,z,ierr)
      if(ierr .ne. 0) go to 50
      call balbak(nm,n,is1,is2,fv1,n,z)
   50 return
      end
c******************** END OF SUBROUTINE RUHESS ************************
```

LISTINGS FOR SUBROUTINES BALANC, HQR, ELTRAN

AND BALBAK ARE CONTAINED IN

THE EISPACK GUIDE (SMITH et al, 1976)

APPENDIX E

FORTRAN LISTING FOR APPROXIMATION SCHEME EMPLOYING THE PUTZER ALGORITHM FOR THE 20x20 TRUNCATION OF THE L MATRIX FOR THE PERTURBED SHO WITH DRAG FORCE

```
$DEBUG
      program ptzdrg20
      parameter(nmax=20,npartshn=5)
      implicit real*8 (a-h,o-z)
      dimension a(nmax,nmax)
      dimension wr(nmax), wi(nmax)
      dimension linefo(6)
      complex*16 ca(nmax,nmax), ceigval(nmax), cident(nmax,nmax)
      complex*16 pfirst(nmax,nmax), psecond(nmax,nmax), temp(2,nmax)
      complex*16 expmatt(2,nmax)
      complex*16 u(nmax), u0(nmax), r_rkgill(nmax), r_last(nmax)
      integer n, nsteps, kpartshn(npartshn)
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      logical doprint /.FALSE./
      logical no_default_ev /.FALSE./
      logical ichgstep /.FALSE./
      logical initialze /.TRUE./
      character*24 filename
      character*24 ev_file
      character*12 linefo
      character*1  idoprint
      character*1  iread_ev_file
      character*1  istepans
      common /matrices/ ca, cident, n
      common /pmatrix/  pfirst, psecond
      data ev_file /'C:\ev54drst.pgm        '/
      data h/.10D+00/
      data fact3/6.0D+00/
c-->     Define partioning structure of the matrix...
c-->  Ending partition indices are given by kpartshn(j) for
c                                              j = 1,...,npartshn
c     Beginning partition indices are given by kpartshn(l)+1 for
c                                          l = 1,...,npartshn-1
      do 450 j = 1, npartshn
        kpartshn(j) = ( (((j+1)*(j+2))/2) - 1)
  450 continue
      write(*,'('' Input the values of m, k, g:'',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YYbSd.dD+ZZ'' )')
      read(*,1229) XM, XK, G
```

```
 1229 format(1x,D8.4,1x,D8.4,1x,D8.4)
      write(*,'('' Enter the value of alpha, the drag coefficient: (in S
     &X.XXXXXD+00) format'')')
      write(*,'('' SX.XXXXXD+00'')')
      read(*,1112) alpha
 1112 format(D12.5)
      write(*,'('' Input the dimension(order) .le.'',I3,'', of the A mat
     &rix.'')' ) kpartshn(npartshn)
      read(*,*) n
      write(*,'('' Input initial values XZERO, and PZERO: '',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YY'' )')
      read(*,1231) xzero, pzero
 1231 format(1x,D8.4,1x,D8.4)
c-->       now get filename to use in writing the output...
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(6,file=filename,status='new')
      write(*,'('' Do you wish to enter the path and filename of the fil
     &e containing the computed eigenvalues? (Y/N)'')')
      read(*,'(A1)') iread_ev_file
      if((iread_ev_file.eq.'Y').or.(iread_ev_file.eq.'y')) no_default_ev
     & = .TRUE.
      if(no_default_ev) then
        write(*,'('' Enter eigenvalue data PATH and FILENAME'')')
        read(*,'(A24)') ev_file
      endif
      open(7,file=ev_file,status='unknown')
      rewind 7
      write(*,'('' Input the init time, end time, and no.of steps.'',/,
     &'' Use the format:'',/,
     &'' D10.4,1X,D10.4,1X,I5'' ,/,
     &'' Sd.dddD+XXbSd.dddD+YYb54321'' )')
      read(*,1232) tbegin, tend, nsteps
 1232 format(1x,D10.4,1x,D10.4,1x,I5)
      write(*,'('' Default value of integration STEPSIZE is:'',D10.4)')
     & h
      write(*,'('' Do you wish to change it? Enter (Y/N)'')')
      read(*,'(A1)') istepans
      if((istepans .eq. 'y') .or. (istepans .eq. 'Y')) ichgstep = .TRUE.
      if(ichgstep) then
        write(*,'(1x,''Enter the value of integration STEPSIZE, use form
     &at'' )')
        write(*,'(1x,''Sd.ddddD+XX'' )')
        read(*,'(D11.4)') h
        write(*,'(D11.4)') h
```

```
      endif
      write(*,'('' Input the time at which to check calculations.'',/,
     &'' Use the format:'',/,
     &'' D11.4'' ,/,
     &'' Sdd.dddD+XX'' )')
      read(*,1233) tchkcalc
 1233 format(1x,D11.4)
      if(tchkcalc .gt. 0.0D+00) doprint = .TRUE.
c-->      Calculate the increment to advance the time with
      dt = (tend - tbegin)/dble(nsteps)
      t0 = tbegin
c-->  CALL GETDAT AND GETTIM (OBTAIN SYSTEM DATE AND TIME)
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
      write(4,260)
      write(4,261) xm, xk, g
      write(4,282) alpha
      write(4,262) xzero, pzero
      write(4,263)
      write(4,264) tbegin, tend, nsteps, dt, h
      write(4,270)
      write(4,265) filename
      write(4,266) imon, iday, iyr,  ihr, iminute, isecond
c         ....write data to output file for records ...
  160 format(A24)
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 format(1H ,' XM IS:',D10.4,'  XK IS: ',D10.4,' G IS: ',D10.4)
  262 format(1H ,' XZERO IS:',D10.4,'  PZERO IS: ',D10.4,' (USED IN BUIL
     &DING INITIAL CONDITION VECTOR)')
  263 format(1H ,' TRANSFORMATION BOUNDARIES FOLLOW: ')
  264 format(1H ,' TBEGIN = ',D11.6,' ; TEND = ',D11.6,' AND NSTEPS = ',
     -I6,/,5x,' DELTAT = ',D11.6,/,5x,' AND RKG STEPSIZE IS: ',D11.6)
  265 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  266 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
  270 format(1H ,' EIGENVALUES PROCESSED TO RUN FROM LARGEST TO SMALLEST
     - WITHIN EACH',/,'   PARTITION OF THE MATRIX.')
  282 format(1H ,' DRAG COEFFICIENT ALPHA IS: ',D12.5)
c-->    Define the A matrix for 20x20 truncation....Drag terms added..
      do 1000 j = 1, n
        do 1010 i = 1, n
          a(i,j) = 0.00D+00
 1010   continue
 1000 continue
c-->   Define L (Upper Hessenberg) matrix for up to n=20 truncation....
      aom = alpha/xm
      a(1,2) = 1.00D+00/xm
      a(2,1) = -xk
```

```
a(2,2) = -aom

a(3,4) = 2.00D+00/xm
a(4,3) = -xk
a(4,4) = -aom
a(4,5) = 1.00D+00/xm
a(5,4) = -2.00D+00*xk
a(5,5) = -2.00D+00*aom

a(2,6) = -g/fact3
a(6,7) =  3.00D+00/xm
a(7,6) = -xk
a(7,7) = -aom
a(7,8) =  2.00D+00/xm
a(8,7) =  -2.00D+00*xk
a(8,8) =  -2.00D+00*aom
a(8,9) = 1.00D+00/xm
a(9,8) = -3.00D+00*xk
a(9,9) = -3.00D+00*aom

a(4,10) = -g/fact3
a(5,11) = -2.00D+00*g/fact3
a(10,11) = 4.00D+00/xm
a(11,10) = -xk
a(11,11) = -aom
a(11,12) =  3.00D+00/xm
a(12,11) = -2.00D+00*xk
a(12,12) = -2.00D+00*aom
a(12,13) =  2.00D+00/xm
a(13,12) = -3.00D+00*xk
a(13,13) = -3.00D+00*aom
a(13,14) =  1.00D+00/xm
a(14,13) = -4.00D+00*xk
a(14,14) = -4.00D+00*aom

a(7,15) =  -g/fact3
a(8,16) =  -2.00D+00*g/fact3
a(9,17) =  -3.00D+00*g/fact3
a(15,16) =  5.00D+00/xm
a(16,15) = -xk
a(16,16) =  -aom
a(16,17) =  4.00D+00/xm
a(17,16) = -2.00D+00*xk
a(17,17) = -2.00D+00*aom
a(17,18) =  3.00D+00/xm
a(18,17) = -3.00D+00*xk
a(18,18) = -3.00D+00*aom
a(18,19) =  2.00D+00/xm
```

```
      a(19,18) = -4.00D+00*xk
      a(19,19) = -4.00D+00*aom
      a(19,20) =  1.00D+00/xm
      a(20,19) = -5.00D+00*xk
      a(20,20) = -5.00D+00*aom

c     a(11,21) = -g/fact3
c     a(12,22) = -2.00D+00*g/fact3
c     a(13,23) = -3.00D+00*g/fact3
c     a(14,24) = -4.00D+00*g/fact3
c     a(21,22) =  6.00D+00/xm
c     a(22,21) = -xk
c     a(22,22) = -aom
c     a(22,23) =  5.00D+00/xm
c     a(23,22) = -2.00D+00*xk
c     a(23,23) = -2.00D+00*aom
c     a(23,24) =  4.00D+00/xm
c     a(24,23) = -3.00D+00*xk
c     a(24,24) = -3.00D+00*aom
c     a(24,25) =  3.00D+00/xm
c     a(25,24) = -4.00D+00*xk
c     a(25,25) = -4.00D+00*aom
c     a(25,26) =  2.00D+00/xm
c     a(26,25) = -5.00D+00*xk
c     a(26,26) = -5.00D+00*aom
c     a(26,27) =  1.00D+00/xm
c     a(27,26) = -6.00D+00*xk
c     a(27,27) = -6.00D+00*aom

c     a(16,28) = -g/fact3
c     a(17,29) = -2.00D+00*g/fact3
c     a(18,30) = -3.00D+00*g/fact3
c     a(19,31) = -4.00D+00*g/fact3
c     a(20,32) = -5.00D+00*g/fact3
c     a(28,29) =  7.00D+00/xm
c     a(29,28) = -xk
c     a(29,29) = -aom
c     a(29,30) =  6.00D+00/xm
c     a(30,29) = -2.00D+00*xk
c     a(30,30) = -2.00D+00*aom
c     a(30,31) =  5.00D+00/xm
c     a(31,30) = -3.00D+00*xk
c     a(31,31) = -3.00D+00*aom
c     a(31,32) =  4.00D+00/xm
c     a(32,31) = -4.00D+00*xk
c     a(32,32) = -4.00D+00*aom
c     a(32,33) =  3.00D+00/xm
c     a(33,32) = -5.00D+00*xk
```

```
c     a(33,33) = -5.00D+00*aom
c     a(33,34) =  2.00D+00/xm
c     a(34,33) = -6.00D+00*xk
c     a(34,34) = -6.00D+00*aom
c     a(34,35) =  1.00D+00/xm
c     a(35,34) = -7.00D+00*xk
c     a(35,35) = -7.00D+00*aom

c     a(22,36) = -g/fact3
c     a(23,37) = -2.00D+00*g/fact3
c     a(24,38) = -3.00D+00*g/fact3
c     a(25,39) = -4.00D+00*g/fact3
c     a(26,40) = -5.00D+00*g/fact3
c     a(27,41) = -6.00D+00*g/fact3
c     a(36,37) =  8.00D+00/xm
c     a(37,36) = -xk
c     a(37,37) = -aom
c     a(37,38) =  7.00D+00/xm
c     a(38,37) = -2.00D+00*xk
c     a(38,38) = -2.00D+00*aom
c     a(38,39) =  6.00D+00/xm
c     a(39,38) = -3.00D+00*xk
c     a(39,39) = -3.00D+00*aom
c     a(39,40) =  5.00D+00/xm
c     a(40,39) = -4.00D+00*xk
c     a(40,40) = -4.00D+00*aom
c     a(40,41) =  4.00D+00/xm
c     a(41,40) = -5.00D+00*xk
c     a(41,41) = -5.00D+00*aom
c     a(41,42) =  3.00D+00/xm
c     a(42,41) = -6.00D+00*xk
c     a(42,42) = -6.00D+00*aom
c     a(42,43) =  2.00D+00/xm
c     a(43,42) = -7.00D+00*xk
c     a(43,43) = -7.00D+00*aom
c     a(43,44) =  1.00D+00/xm
c     a(44,43) = -8.00D+00*xk
c     a(44,44) = -8.00D+00*aom

c     a(29,45) = -g/fact3
c     a(30,46) = -2.00D+00*g/fact3
c     a(31,47) = -3.00D+00*g/fact3
c     a(32,48) = -4.00D+00*g/fact3
c     a(33,49) = -5.00D+00*g/fact3
c     a(34,50) = -6.00D+00*g/fact3
c     a(35,51) = -7.00D+00*g/fact3
c     a(45,46) =  9.00D+00/xm
c     a(46,45) = -xk
```

```
c     a(46,46) = -aom
c     a(46,47) =  8.00D+00/xm
c     a(47,46) = -2.00D+00*xk
c     a(47,47) = -2.00D+00*aom
c     a(47,48) =  7.00D+00/xm
c     a(48,47) = -3.00D+00*xk
c     a(48,48) = -3.00D+00*aom
c     a(48,49) =  6.00D+00/xm
c     a(49,48) = -4.00D+00*xk
c     a(49,49) = -4.00D+00*aom
c     a(49,50) =  5.00D+00/xm
c     a(50,49) = -5.00D+00*xk
c     a(50,50) = -5.00D+00*aom
c     a(50,51) =  4.00D+00/xm
c     a(51,50) = -6.00D+00*xk
c     a(51,51) = -6.00D+00*aom
c     a(51,52) =  3.00D+00/xm
c     a(52,51) = -7.00D+00*xk
c     a(52,52) = -7.00D+00*aom
c     a(52,53) =  2.00D+00/xm
c     a(53,52) = -8.00D+00*xk
c     a(53,53) = -8.00D+00*aom
c     a(53,54) =  1.00D+00/xm
c     a(54,53) = -9.00D+00*xk
c     a(54,54) = -9.00D+00*aom

c     a(37,55) = -g/fact3
c     a(38,56) = -2.00D+00*g/fact3
c     a(39,57) = -3.00D+00*g/fact3
c     a(40,58) = -4.00D+00*g/fact3
c     a(41,59) = -5.00D+00*g/fact3
c     a(42,60) = -6.00D+00*g/fact3
c     a(43,61) = -7.00D+00*g/fact3
c     a(44,62) = -8.00D+00*g/fact3
c     a(55,56) =  10.00D+00/xm
c     a(56,55) = -xk
c     a(56,56) = -aom
c     a(56,57) =  9.00D+00/xm
c     a(57,56) = -2.00D+00*xk
c     a(57,57) = -2.00D+00*aom
c     a(57,58) =  8.00D+00/xm
c     a(58,57) = -3.00D+00*xk
c     a(58,58) = -3.00D+00*aom
c     a(58,59) =  7.00D+00/xm
c     a(59,58) = -4.00D+00*xk
c     a(59,59) = -4.00D+00*aom
c     a(59,60) =  6.00D+00/xm
c     a(60,59) = -5.00D+00*xk
```

```
c     a(60,60) = -5.00D+00*aom
c     a(60,61) =  5.00D+00/xm
c     a(61,60) = -6.00D+00*xk
c     a(61,61) = -6.00D+00*aom
c     a(61,62) =  4.00D+00/xm
c     a(62,61) = -7.00D+00*xk
c     a(62,62) = -7.00D+00*aom
c     a(62,63) =  3.00D+00/xm
c     a(63,62) = -8.00D+00*xk
c     a(63,63) = -8.00D+00*aom
c     a(63,64) =  2.00D+00/xm
c     a(64,63) = -9.00D+00*xk
c     a(64,64) = -9.00D+00*aom
c     a(64,65) =  1.00D+00/xm
c     a(65,64) = -10.00D+00*xk
c     a(65,65) = -10.00D+00*aom
c-->     Write out A matrix....
      write(*,'('' Portion of A matrix used is:  '')' )
      write(4,'('' Portion of A matrix used is:  '')' )
      do 1180 k = 1, n
        write(*,'('' row '',i3)') k
        write(*,'('' '',6(D11.5,1x),/)' ) (a(k,j), j = 1,n)
 1180 continue
      do 1190 k = 1, n
        write(4,'('' row '',i3)') k
        write(4,'('' '',6(D11.5,1x),/)' ) (a(k,j), j = 1,n)
 1190 continue
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
c-->    convert matrix A into matrix CA, its complex form
      do 1230 jj = 1, n
        do 1240 ii = 1, n
          ca(ii,jj) = dcmplx( a(ii,jj), 0.00D+00 )
 1240   continue
 1230 continue
c-->    Form the vector of complex initial valued basis parameters
      u0(1) = dcmplx(xzero, 0.00D+00 )
      u0(2) = dcmplx(pzero, 0.00D+00 )
      u0(3) = dcmplx(xzero*xzero, 0.00D+00 )
      u0(4) = dcmplx(xzero*pzero, 0.00D+00 )
      u0(5) = dcmplx(pzero*pzero, 0.00D+00 )
      u0(6) = dcmplx(xzero*xzero*xzero, 0.00D+00 )
      u0(7) = dcmplx(xzero*xzero*pzero, 0.00D+00 )
      u0(8) = dcmplx(xzero*pzero*pzero, 0.00D+00 )
      u0(9) = dcmplx(pzero*pzero*pzero, 0.00D+00 )
      u0(10) = dcmplx(xzero*xzero*xzero*xzero, 0.00D+00 )
      u0(11) = dcmplx(xzero*xzero*xzero*pzero, 0.00D+00 )
      u0(12) = dcmplx(xzero*xzero*pzero*pzero, 0.00D+00 )
```

```
      u0(13) = dcmplx(xzero*pzero*pzero*pzero, 0.00D+00 )
      u0(14) = dcmplx(pzero*pzero*pzero*pzero, 0.00D+00 )
      u0(15) = dcmplx(xzero*xzero*xzero*xzero*xzero, 0.00D+00 )
      u0(16) = dcmplx(xzero*xzero*xzero*xzero*pzero, 0.00D+00 )
      u0(17) = dcmplx(xzero*xzero*xzero*pzero*pzero, 0.00D+00 )
      u0(18) = dcmplx(xzero*xzero*pzero*pzero*pzero, 0.00D+00 )
      u0(19) = dcmplx(xzero*pzero*pzero*pzero*pzero, 0.00D+00 )
      u0(20) = dcmplx(pzero*pzero*pzero*pzero*pzero, 0.00D+00 )
c-->    Build complex identity matrix of order n
      do 1250 jj = 1, n
        do 1260 ii = 1, n
          if(jj .eq. ii) then
            cident(ii,ii) = dcmplx(1.0D+00, 0.0D+00)
          else
            cident(ii,jj) = dcmplx(0.0D+00, 0.0D+00)
          endif
 1260   continue
 1250 continue
c--> ######################################################################
c       Read in the eigenvalues from the file EV54DRST.PGM.(default) or
c         from previously entered file.
c         First read past header containing info on physical constants
c         used to obtain the set of eigenvalues, and file creation info.
c
c-->      Read file until the header is passed.  The header is delimited
c-->      by a line that consists of the character string,
c-->      +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/
c-->      as a delimiter.
c-->      read to get past header on file known as 'ev_file'.
 1111 read(7,'(6(A12))',end=215) linefo
      if((linefo(2) .eq. '+/+/+/+/+/+/') .or. (linefo(2) .eq.
     $ '/+/+/+/+/+/+') ) then
        go to 500
      else
        go to 1111
      endif
c-->     re-set linefo.....
  500 do 335 ik = 1,6
        linefo(ik) = '            '
  335 continue
c--> we will open the file and read in the eigenvalues for the current
c-->      truncation of the L matrix.
      do 10 i = 1, n
        read(7,'(D16.10,2x,D16.10)',end=225) wr(i), wi(i)
   10   continue
c     ***** Form complex eigenvalue vector ......
        do 300 i = 1, n
          ceigval(i) = dcmplx( wr(i),wi(i) )
```

```
 300   continue
       if(doprint) then
         write(*,'('' Complex eigenvalue vector follows: '')')
         write(4,'('' Complex eigenvalue vector follows: '')')
         do 325 i = 1, n
           if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &       (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) .or.
     &       (i.eq.(kpartshn(5)+1)) )
     &       write(*,'(''     '')')
           if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &       (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) .or.
     &       (i.eq.(kpartshn(5)+1)) )
     &     write(4,'(''     '')')
           write(*,'(''  i = '',i3,'' EIGENVALUE: '',D14.8,1x,D14.8,''(
     &I)'')' ) i, wr(i), wi(i)
           write(4,'(''  i = '',i3,'' EIGENVALUE: '',D14.8,1x,D14.8,''(
     &I)'')' ) i, wr(i), wi(i)
 325     continue
       endif
c        Call routine BuildP to build two storage areas.  The first
c          PFIRST contains the first row of the P matrix having indice
c          INDEX, e.g. PFIRST(INDEX,row). The second, PSECOND, contains
c          the second row of the P matrix having indice INDEX, e.g.
c          PSECOND(INDEX,row).
     call buildp20(ceigval)
c-->     Write out the PFIRST & PSECOND matrices, if requested ......
     if(doprint) then
       write(*,'('' PFIRST & PSECOND Matrices follow:   '')' )
       write(4,'('' PFIRST & PSECOND Matrices follow:   '')' )
       do 3170 k = 1, n
         write(*,'('' P matrix index is '',i3)') k
           write(*,'('' row 1 '')')
           write(*,'('' '',3(2(G11.4,1x),1x),/)') (pfirst(k,j), j= 1,n)
           write(*,'('' row 2 '')')
           write(*,'('' '',3(2(G11.4,1x),1x),/)') (psecond(k,j),j= 1,n)
           write(*,'(1x,70(1H-))')
3170   continue
       do 4170 k = 1, n
         write(4,'('' P matrix index is '',i3)') k
           write(4,'('' row 1 '')')
           write(4,'('' '',3(2(G11.4,1x),1x),/)') (pfirst(k,j), j= 1,n)
           write(4,'('' row 2 '')')
           write(4,'('' '',3(2(G11.4,1x),1x),/)') (psecond(k,j),j= 1,n)
           write(4,'(1x,70(1H-))')
4170   continue
       write(*,'('' '',71(1H-))' )
       write(4,'('' '',71(1H-))' )
     endif
```

```
c     ************ BEGIN ITERATING IN TIME **********************
      write(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
      write(4,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
      do 40 k=1,nsteps + 1
        tlast = t
        if((t .eq. 0.0D+00) .and. (initialze)) then
          r_rkgill(1) = dcmplx(1.0D+00, 0.0D+00)
          do 8200 kl = 2, n
            r_rkgill(kl) = dcmplx(0.0D+00, 0.0D+00)
 8200     continue
          initialze = .FALSE.
        else
c       Define last values of the r_rkgill(n) vector for use as the
c       initial condition vector for the RKGILL integration.
          do 8300 kn = 1, n
            r_last(kn) = r_rkgill(kn)
 8300     continue
c-->    Get the entire set of polynomial functions, each evaluated at the
c         integrated time T, given that the set of polynomials have the
c         initial conditions given by the values contained within the
c         the vector r_last(n).....
c
c       Pass DT to determine the number of times to execute the Runge-
c         Kutta-Gill integrator routine...
          call getrk20(h,n,ceigval,dt,r_last,r_rkgill,tlast,t)
c-----------------------------------------------------------------------
        endif
c       Initialize array to hold results of this particular time.
c          (iteration of deltat)
        do 3050 ii = 1, 2
          do 3060 jj = 1, n
            expmatt(ii,jj) = dcmplx(0.0D+00,0.0D+00)
 3060     continue
 3050   continue
        if((doprint) .and. (dabs(t - tchkcalc) .le. .02D+00)) then
          do 8400 kn = 1, n
            write(*,'('' For r sub '',i3,'', r is: '',D14.8,2x,D14.8)')
     &        kn, r_rkgill(kn)
 8400     continue
          do 8500 kn = 1, n
            write(4,'('' For r sub '',i3,'', r is: '',D14.8,2x,D14.8)')
     &        kn, r_rkgill(kn)
 8500     continue
        endif
c-->      Calculate the terms in the Putzer sum, accumulating the sum
c           continuously
        do 30000 index = 1, n
          do 3100 jj= 1, n
```

```
            temp(1,jj) =  r_rkgill(index)*pfirst(index,jj)
            temp(2,jj) =  r_rkgill(index)*psecond(index,jj)
 3100      continue
c         **** Calculate the term of the Putzer sum depending on index.
c         **** Then add to the first index-1 sums stored in expmatt.
          do 3200 ii = 1, 2
            do 3210 jj = 1, n
              expmatt(ii,jj) = expmatt(ii,jj) + temp(ii,jj)
 3210       continue
 3200     continue
30000   continue
c--> Initialize the u vector to zero.....
        do 50000 i = 1, n
          u(i) = dcmplx( 0.0D+00, 0.0D+00)
50000   continue
c--> Form the coordinate vector for time t
        do 90100 i = 1, 2
          do 90110 j = 1, n
            u(i) = u(i) + expmatt(i,j)*u0(j)
90110     continue
90100   continue
        x    = dreal(u(1))
        pmom = dreal(u(2))
        write(4,'(1x,D11.4,3x,D17.9,2x,D17.9)' ) t, x, pmom
   40 continue
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
      close(4)
      go to 99999
  215 write(*,'('' EOF found in header of eigenvalue data file: '',A24)
     &') ev_file
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
      close(4)
      stop 00215
  225 write(*,'('' EOF found while reading eigenvalue data on file: '',
     &A24)') ev_file
      write(*,'('' '',71(1H*))' )
      write(4,'('' '',71(1H*))' )
      close(4)
      stop 00225
99999 continue
      end
c******************** END of PROGRAM PTZDRG20 ***************************
```

LISTINGS FOR SUBROUTINES CALLED ARE CONTAINE

APPENDIX B

APPENDIX F

FORTRAN LISTING FOR "EXACT" RUNGE-KUTTA-GILL NUMERICAL INTEGRATION OF THE 20x20 TRUNCATION OF THE L MATRIX FOR THE PERTURBED SHO WITH DRAG FORCE

```
$DEBUG
      PROGRAM SHAMDRAG
C
C*************************************************************************
C*************************************************************************
C********                                                          ********
C********  This program solves a specified Hamiltonian by integrat- ********
C********    ing the resultant Hamilton's equations using a general-********
C********    ized fourth order RUNGE-KUTTA technique employing      ********
C********    GILL'S method. ***********************************************
C********    The Runge-Kutta routine for N simultaneous linear      ********
C********    first-order equations is taken from Appendix C of the  ********
C********    book VISCOUS FLUID FLOW by FRANK M. WHITE 1974 McGraw- ********
C********    Hill pp. 675-678 which implements GILL'S method of     ********
C********    Runge-Kutta integration. *****************************************
C********  The program prompts the user for a filename (DOS) to     ********
C********    which it will write the data.  As written, at most     ********
C********    500 data points will be written.                       ********
C********                                                          ********
C********    H =  P*P/(2*M) + .5 K X*X + G*X*X*X*X/4!              ********
C********    Drag term = F = -(ALPHA*P*P)/(2*(M*M))                ********
C********                  f = -ALPHA*P/M                          ********
C********                                                          ********
C*************************************************************************
C*************************************************************************
C
      PARAMETER(NMAX=2)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER*2 IHR,IMINUTE,ISECOND,I100TH,IYR,IMON,IDAY
      LOGICAL ICH6STEP /.FALSE./
      LOGICAL INITIALIZE /.TRUE./
      CHARACTER*24 FILENAME
      CHARACTER*1  ISTEPANS
      DIMENSION  Y(2), Y_LAST(2)
      DATA H/.10D+00/
C-->  CALL GETDAT AND GETTIM (OBTAIN SYSTEM DATE AND TIME)
      CALL GETDAT (IYR,IMON,IDAY)
      CALL GETTIM (IHR,IMINUTE,ISECOND,I100TH)
C          ....READ SYSTEM DATA
      WRITE(*,'('' Input the values of m, k, g:'',/,
     &'' Use the format:'',/,
```

```
     &'' D8.4,1X,D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YYbSd.dD+ZZ'' )')
      READ(*,1230) XM, XK, G
 1230 FORMAT(1X,D8.4,1X,D8.4,1X,D8.4)
      WRITE(*,'('' ENTER THE VALUE OF ALPHA, THE DRAG COEFFICIENT: (IN S
     &X.XXXXXD+00) FORMAT'')')
      WRITE(*,'('' SX.XXXXXD+00'')')
      READ(*,1112) ALPHA
 1112 FORMAT(D12.5)
      WRITE(*,'('' Input initial values XZERO, and PZERO: '',/,
     &'' Use the format:'',/,
     &'' D8.4,1X,D8.4'' ,/,
     &'' Sd.dD+XXbSd.dD+YY'' )')
      READ(*,1231) XZERO, PZERO
 1231 FORMAT(1X,D8.4,1X,D8.4)
      WRITE(*,'('' Input the init time, end time, and no.of steps.'',/,
     &'' Use the format:'',/,
     &'' D10.4,1X,D10.4,1X,I5'' ,/,
     &'' Sd.dddD+XXbSd.dddD+YYb54321'' )')
      READ(*,1232) TBEGIN, TEND, NSTEPS
 1232 FORMAT(1X,D10.4,1X,D10.4,1X,I5)
      WRITE(*,'('' Default value of integration STEPSIZE is:'',D10.4)')
     & H
      WRITE(*,'('' Do you wish to change it? Enter (Y/N)'')')
      READ(*,'(A1)') ISTEPANS
      IF((ISTEPANS .EQ. 'y') .OR. (ISTEPANS .EQ. 'Y')) ICHGSTEP = .TRUE.
      IF(ICHGSTEP) THEN
        WRITE(*,'(1x,''Enter the value of integration STEPSIZE, use form
     &at'' )')
        WRITE(*,'(1x,''Sd.ddddD+XX'' )')
        READ(*,'(D11.4)') H
        WRITE(*,'(D11.4)') H
      ENDIF
C-->       CALCULATE THE INCREMENT TO ADVANCE THE TIME WITH
      DELTAT = (TEND - TBEGIN)/DBLE(NSTEPS)
      TO = TBEGIN
C-->  Initialize Runge routine variables, control index (M)
      M = 0
C-->        now get filename, starting time and time step for x and p
      WRITE(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      READ(*,'(A24)') FILENAME
C-->     ...OPEN FILE TO SAVE INTEGRATED VALUES OF Y(1), AND Y(2) (X,P)
      OPEN(6,FILE=FILENAME,STATUS='NEW')
C         ....write data to output file for records ...
  259 FORMAT(1H ,'RUNNING 4TH ORDER RUNGE-KUTTA GILL INTEGRATION ON THE
     $DATA BELOW.')
  260 FORMAT(1H ,'SUMMARY OF DATA TO BE USED FOLLOWS:')
```

```
  261 FORMAT(1H ,'XM IS:',D10.4,'  XK IS: ',D10.4,' G IS: ',D10.4)
  262 FORMAT(1H ,'XZERO IS: ',D10.4,' ; PZERO IS: ',D10.4)
  263 FORMAT(1H ,'INTEGRATION LIMITS FOLLOW;        NSTEPS IS ',I6)
  264 FORMAT(1H ,'TBEGIN = ',D11.6,' ; TEND = ',D11.6,' DELTAT = ',
     - D11.6,/,5X,' AND RKG STEPSIZE IS: ',D11.6)
  265 FORMAT(1H ,'OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  266 FORMAT(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
  282 FORMAT(1H ,' DRAG COEFFICIENT ALPHA IS: ',D12.5)
C        ....WRITE DATA TO SCREEN FOR VERIFICATION ...
      WRITE(*,259)
      WRITE(*,260)
      WRITE(*,261) XM, XK, G
      WRITE(*,282) ALPHA
      WRITE(*,262) XZERO, PZERO
      WRITE(*,263) NSTEPS
      WRITE(*,264) TBEGIN, TEND, DELTAT, H
      WRITE(*,265) FILENAME
      WRITE(*,266) IMON, IDAY, IYR,  IHR, IMINUTE, ISECOND
      WRITE(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
C-->    WRITE DATA TO OUTPUT FILE...
      WRITE(6,259)
      WRITE(6,260)
      WRITE(6,261) XM, XK, G
      WRITE(6,282) ALPHA
      WRITE(6,262) XZERO, PZERO
      WRITE(6,263) NSTEPS
      WRITE(6,264) TBEGIN, TEND, DELTAT, H
      WRITE(6,265) FILENAME
      WRITE(6,266) IMON, IDAY, IYR,  IHR, IMINUTE, ISECOND
      WRITE(6,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
C         .....INITIALIZE NECESSARY VARIABLES
      N = NMAX
      T0 = 0.0
      T = T0
C-----------------------------------------------------------------------
      DO 44 K=1, NSTEPS + 1
        T_LAST = T
        IF((T .EQ. 0.0) .AND. (INITIALZE)) THEN
          Y(1) = XZERO
          Y(2) = PZERO
          INITIALZE = .FALSE.
        ELSE
C--> Define last values of the Y(N) vector for use as the
C       initial condition vector for the RKGill integration.
          Y_LAST(1) = Y(1)
          Y_LAST(2) = Y(2)
C--> For the time T, get the entire set of the Y(n)
```

```
C        functions, each evaluated at the time T, given that the set
C        has the initial conditions given by the
C        values contained within the vector Y_LAST(n).....
C
C     Pass DT to determine the number of times to execute Runge-Kutta-
C        Gill routine
         CALL DORKG(N,XM,XK,G,ALPHA,DELTAT,H,Y_LAST,Y,T_LAST,T)
C-----------------------------------------------------------------------
       ENDIF
C-->  Write values to the output file.....
       WRITE(6,'(1x,D12.6,4x,D15.9,4x,D15.9)' ) T, Y(1), Y(2)
   44 CONTINUE
C       ....CLOSE OUTPUT FILE.....
      WRITE(*,'('' '',71(1H*))' )
      WRITE(6,'('' '',71(1H*))' )
      ENDFILE 6
      REWIND 6
      CLOSE(6)
      END
C************************** END OF SHAMDRAG ************************
$DEBUG
      SUBROUTINE DORKG(N,XM,XK,G,ALPHA,DT,H,Y_BEGIN,Y,T_BEGIN,T_END)
C***********************************************************************
C*******                                                          ***
C******* THIS ROUTINE SERVES AS THE INTERFACE TO THE RUNGE-KUTTA-GILL***
C*******  SUBROUTINE rkgill.  IT CONTROLS THE NUMBER OF TIMES THAT   ***
C*******  THE RUNGE STEPS OF rkgill ARE EXECUTED. IT IS DESINED TO   ***
C*******  ADVANCE THE SOLUTION OF Y AT TIME T_LAST, Y_LAST, TO THE   ***
C*******  THE SOLUTION VALID AT TIME T, GIVEN BY Y. ******************
C***********************************************************************
C***********************************************************************
      PARAMETER(NMAX=2)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER M, N, KONTROL
      INTEGER DELTAT_STEPS, KNT_STEPS
      DIMENSION F(2), Y(2), Y_BEGIN(2)
      DATA FACT3/6.0D+0/, FACT4/24.0D+0/
C-->  Determine the total number of steps (DELTAT_STEPS) to make with
C       Runge-Kutta-Gill routine executing stepsize of H to advance
C       solution by amount DELTAT, from T_BEGIN to T.....
      YY = DT/H
      IF(YY .LT. 1.0) STOP 00111
      DTT = DT
      KLIM = IDINT(YY)
C       Do loop used to get precise correct value for DELTAT_STEPS
      DO 4545 JH = 1, KLIM + 2
        DTT = DTT - H
        IF((DTT - 0.0D+00) .LE. 1.0D-06) GO TO 4550
```

```
 4545 CONTINUE
 4550 DELTAT_STEPS = JH
C--)  Initialize Runge routine variables, control index (m)
      M = 0
C--)  Initialize step counter (knt_steps)......
      KNT_STEPS = 0
      T = T_BEGIN
      Y(1) = Y_BEGIN(1)
      Y(2) = Y_BEGIN(2)
C-------------------------------------------------------------------
C
C       ....CALL ON 4-TH ORDER RUNGE-KUTTA-GILLS FUNCTION
    8 IF(KNT_STEPS - DELTAT_STEPS) 6,7,7
    6 CONTINUE
      CALL RKGILL(N,Y,F,T,H,M,KONTROL)
      GO TO (10,20) KONTROL
c       Define first-order derivatives of the system....
   10 F(1) = Y(2)/XM
      F(2) = -(XK + (G*Y(1)*Y(1))/FACT3)*Y(1) - ALPHA*Y(2)/XM
      GO TO 6
   20 CONTINUE
      KNT_STEPS = KNT_STEPS + 1
      GO TO 8
    7 CONTINUE
      T_END = T
      RETURN
      END
C********************  END OF DORKG ***********************************
$DEBUG
      subroutine rkgill(n,y,f,x,h,m,kontrl)
c*************************************************************************
c*******  This subroutine performs Runge-Kutta integration by the Gills*
c*******  method.  The Runge-Kutta routine for N ************************
c*******  simultaneous linear first-order equations is taken from *******
c*******  Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c*******  1974 McGraw-Hill pp. 675-678  *********************************
c*************************************************************************
      parameter(nmax=2)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrl
      dimension  y(nmax), f(nmax), q(nmax)
      m = m + 1
      go to (1,4,5,3,7) m
    1 do 2 i = 1, n
        q(i) = 0.0D+00
    2 continue
      a = 0.5D+00
      go to 9
```

```
    3 a = 1.7071067811865475244D+00
    4 x = x + (.5D+00)*h
    5 do 6 i = 1, n
        y(i) = y(i) + a*(f(i)*h - q(i))
        q(i) = (2.0D+00)*a*h*f(i) + ((1.0D+00) - (3.0D+00)*a)*q(i)
    6 continue
      a = .29289321881345247560+00
      go to 9
    7 do 8 i = 1, n
        y(i) = y(i) + h*f(i)/(6.0D+00)  -  q(i)/(3.0D+00)
    8 continue
      m = 0
      kontrl = 2
      go to 10
    9 kontrl = 1
   10 continue
      return
      end
c****************  END OF SUBROUTINE RKGILL  ***************************
```

APPENDIX G

FORTRAN LISTING FOR EIGENVALUE CALCULATION OF 10x10 L MATRIX FOR KEPLER PROBLEM

```
$DEBUG
      program evkep110
c**************************************************************************
c**************************************************************************
c*******  This program is a driver program for use with the EISPACK ******
c*******  guide.  The intent of the program is to calculate eigenvalues*
c*******  only.  It sets up the matrix to be used by the subsequent ******
c*******  driver subroutine RG.  All routines are taken from the    ******
c*******  EISPACK guide.  If no external radial force is applied the ******
c*******  matrix reduces to an upper Hessenberg matrix. This program ******
c*******  obtains the eigenvalues for the 10x10 truncation of the   ******
c*******  matrix.  The matrix is partitioned to obtain its general  ******
c*******  structure with respect to the polynomial basis.  Logic to ******
c*******  order the eigenvalues within each partition from largest to******
c*******  smallest in the direction of increasing index is installed.******
c*******  The execution of this logic is at the option of the user.  ******
c*******  The resultant eigenvalue information is written to a data  ******
c*******  file capable of being read by an external program, or to a ******
c*******  file which can be printed out. ********************************
c*******  In either case a header containing information of physical ******
c*******  constants and units used is placed at the top of each file.******
c**************************************************************************
c********                                                           ******
c********  This program solves for the eigenvalues of the L matrix  ******
c********    produced by the action of the Liouville operator upon  ******
c********    a basis consisting of    phi, and polynomials consisting *
c********    of q, the radial deviation from an equilbrium circular ******
c********    orbit, and p, the corresponding conjugate momenta. The ******
c********    Hamiltonian used is that for the Kepler problem in a   ******
c********    plane.  Two sets of units are provided in which the    ******
c********    problem can be worked.  They are the physical units    ******
c********    provided by the MKS system, or the canonical units     ******
c********    determined in such a way that GM = 1. *********************
c********  The program prompts the user for a filename (DOS) to     ******
c********     which it will write the eigenvalue data.              ******
c********  The Hamiltonian is as follows: ********************************
c********                                                           ******
c********  In MKS units...                                          ******
c********     H =  (Pr)*(Pr)/(2m*r0*r0) + ((lphi)**2)/(2*m*r0*r0*   ******
c********          (1 + Q)**2) - GMm/(r0*(1+Q))                     ******
c********                                                           ******
```

```
c********   In CANONICAL units....                                        ***
c********      H =  (Pr)*(Pr)/(2*r0*r0) + ((lphi)**2)/(2*r0*r0*          ***
c********           (1 + Q)**2) -   1/(r0*(1+Q))                         ***
c********                                                                ***
c********   The user is prompted for his choice of units to work in.     ***
c********                                                                ***
c*************************************************************************
c*******  Units are in the MKS system. Kg, Km, sec OR a CANONICAL UNIT *
c*******  system Kg, Radius of earth in Km, CTU (Canonical Time Unit) **
c*******  = 806.5 sec = 13.44 min, at the choice of the user. **********
c*************************************************************************
c*******  Cylindrical coordinate system is assumed.  r, phi, z ********
c*************************************************************************
c*************************************************************************
      parameter(nmax=10,npartshn=4)
      implicit real*8 (a-h,o-z)
      real*8 a(nmax,nmax), z(nmax,nmax), wr(nmax), wi(nmax), fv1(nmax)
      real*8 xm, xk, g
c     real zr(nmax,nmax), zi(nmax,nmax)
      dimension rinit(3), vinit(3), anominit(3), c(3)
      logical canonical /.FALSE./
      logical ichgstep /.FALSE./
      logical idoprint /.FALSE./
      logical lswap /.FALSE./
      logical dosort /.FALSE./
      logical initialze /.TRUE./
      integer n, nm, matz, iv1(nmax), ierr, kpartshn(npartshn)
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      character*24 filename
      character*24 filename1
      character*1  manswer
      character*1  idosort
      character*1 i_pick_units
      character*8 GMunits
      character*2 Runits
      character*3 Tunits
      character*6 Vunits
      character*9 Punits
      character*10 AMunits
c-->     Physical constant data....
      data convmkm/1.0D-03/
      data pi/3.1415927D+00/
      data G/6.67D-17/, earth_mas/5.98D+24/, r_earth/6.378165D+03/
      data ctu_per_sec/1.23985210-03/
c--> Call GETDAT and GETTIM (obtain system date and time)
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
c-->     Define partioning structure of the matrix...
```

```
c--)  Ending partition indices are given by kpartshn(j) for
c                                                  j = 1,...,npartshn
c     Beginning partition indices are given by kpartshn(l)+1 for
c                                               l = 1,...,npartshn-1
      do 450 j = 1, npartshn
        kpartshn(j) = ( (j*(j+1))/2 )
  450 continue
      write(*,'('' Input the dimension(order) .le.'',I3,'', of the A mat
     &rix.'')' ) kpartshn(npartshn)
      read(*,*) n
      write(*,'('' Do you want the matrix printed out? y or n '')')
      read(*,130) manswer
      if((manswer .eq. 'y' ) .or. (manswer .eq. 'Y')) idoprint = .TRUE.
      write(*,'('' Do you want the eigenvalues sorted in monotonic decr
     & order? (y or n) '')')
      read(*,130) idosort
      if((idosort .eq. 'y' ) .or. (idosort .eq. 'Y')) dosort = .TRUE.
c--)        now get filename to use in writing the output...
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(6,file=filename,status='new')
c--)        now get filename to write eigenvalues to be read by another
c             program.......
      write(*,'('' Please enter the filename to use to write data for pr
     -ogram accessability.'')' )
      read(*,160) filename1
      open(5,file=filename1,status='new')
      write(*,'('' Input the value of n in Kg:'',/,
     &'' Use the format:'',/,
     &'' D11.4'' ,/,
     &'' Sd.dddd0+XX'' )')
      read(*,1229) xm
 1229 format(1x,D11.4)
      write(*,'('' Default coord system units is the MKS system.'')')
      write(*,'('' Do you wish to use CANONICAL system of units? Enter
     &Y or N'')')
      read(*,'(A1)') i_pick_units
      if((i_pick_units .eq. 'Y') .or. (i_pick_units .eq. 'y')) canonical
     & =  .TRUE.
c--)    Input initial components for position vector....
      write(*,'('' Input initial values RINITIAL(radial) in Km: '',/,
     &'' Use the format:'',/,
     &'' D14.7'' ,/,
     &'' Sd.ddddddd0+XX'' )')
      read(*,1231) rinit_radial
 1231 format(1x,D14.7)
 1232 format(1x,D14.7)
```

```
      write(*,1232) rinit_radial
      if( .not. (canonical) ) then
        GM = G*earth_mas*convmkm
        rinit(1) = rinit_radial
        rinit(2) = 0.0D+00
        rinit(3) = 0.0D+00
      else
        GM = 1.0D+00
        xm = 1.0D+00
        rinit(1) = rinit_radial/r_earth
        rinit(2) = 0.0D+00
        rinit(3) = 0.0D+00
      endif
c-->    Input initial components for momentum vector....
      write(*,'('' Input initial velocity values VINITIAL(radial), and V
     &INITIAL(transverse): in Km/sec '',/,
     &'' Use the format:'',/,
     &'' D14.7,1X,D14.7'' ,/,
     &'' Sd.dddddddD+XXbSd.dddddddD+YY'' )')
      read(*,1233) vinit_radial, vinit_trans
      write(*,1234) vinit_radial, vinit_trans
 1233 format(1x,D14.7,1x,D14.7)
 1234 format(1x,D14.7,1x,D14.7)
      if( .not. (canonical) ) then
        vinit(1) = vinit_radial
        vinit(2) = vinit_trans
        vinit(3) = 0.0D+00
      else
        vinit(1) = (vinit_radial/r_earth)/ctu_per_sec
        vinit(2) = (vinit_trans/r_earth)/ctu_per_sec
        vinit(3) = 0.0D+00
      endif
c-->        Calculation of initial angular momentum....
      call crosspr(rinit,vinit,c)
      amominit(1) = xm*c(1)
      amominit(2) = xm*c(2)
      amominit(3) = xm*c(3)
c-->        Calculation of magnitude of initial angular momentum....
      xlphisq = dot(amominit,amominit)
      xlphi = dsqrt(xlphisq)
c-->        Calculation of circular orbit radius....
      if( .not. (canonical) ) then
        r0 = xlphisq/(GM*xm*xm)
        r0overre = r0/r_earth
      else
        r0 = xlphisq
        r0overre = r0
      endif
```

```
        aa  = r0
c-->       Calculation of initial period using Kepler's Third Law
      if( .not. (canonical) ) then
        period = (2.0D+00*pi*aa*dsqrt(aa))/dsqrt(GM)
        period_min = period/60.0D+00
      else
        period = 2.0D+00*pi*aa*dsqrt(aa)
        period_min = period*(1.0D+00/ctu_per_sec)*(1.0D+00/60.0D+00)
      endif
c-->    Calculate the constants used in the matrix of the L matrix.....
      if( .not. (canonical) ) then
        alpha = 1.0D+00/(xm*r0*r0)
        alphasq = alpha*alpha
        beta  = GM*xm/r0
      else
        alpha = 1.0D+00/(r0*r0)
        alphasq = alpha*alpha
        beta = 1.0D+00/r0
      endif
c         ....write data to output file for records ...
      if( .not. (canonical) ) then
        GMunits = 'Km3/sec2'
        Runits  = 'Km'
        Vunits  = 'Km/sec'
        Tunits  = 'sec'
        Punits  = 'Kg-Km/sec'
        AMunits = 'Kg-Km2/sec'
      else
        GMunits = 'Re3/CTU2'
        Runits  = 'Re'
        Vunits  = 'Re/CTU'
        Tunits  = 'CTU'
        Punits  = 'Kg-Re/CTU'
        AMunits = 'Kg-Re2/CTU'
      endif
      write(6,259)
      write(6,258)
      write(6,255)
      write(6,250)
      write(6,260)
      write(6,261) xm, GM, GMunits
      write(6,262) rinit(1), Runits
      write(6,263) vinit(1), Vunits, vinit(2), Vunits
      write(6,250)
      write(6,265) (amominit(i), i = 1,3)
      write(6,266) xlphisq, AMunits, xlphi, AMunits
      write(6,267) r0, Runits, period, Tunits, period_min
      write(6,268) r0overre
```

```
      write(6,250)
      if(dosort) write(6,270)
      write(6,272) filename
      write(6,273) imon, iday, iyr,  ihr, iminute, isecond
C       write data to file to be used by external program...
      write(5,259)
      write(5,258)
      write(5,255)
      write(5,250)
      write(5,260)
      write(5,261) xm, GM, GMunits
      write(5,262) rinit(1), Runits
      write(5,263) vinit(1), Vunits, vinit(2), Vunits
      write(5,250)
      write(5,265) (amominit(i), i = 1,3)
      write(5,266) xlphisq, AMunits, xlphi, AMunits
      write(5,267) r0, Runits, period, Tunits, period_min
      write(5,268) r0overre
      write(5,250)
      if(dosort) write(5,270)
      write(5,272) filenamei
      write(5,273) imon, iday, iyr,  ihr, iminute, isecond
C         ....write data to screen for verification ...
      write(*,259)
      write(*,258)
      write(*,255)
      write(*,250)
      write(*,260)
      write(*,261) xm, GM, GMunits
      write(*,262) rinit(1), Runits
      write(*,263) vinit(1), Vunits, vinit(2), Vunits
      write(*,250)
      write(*,265) (amominit(i), i = 1,3)
      write(*,266) xlphisq, AMunits, xlphi, AMunits
      write(*,267) r0, Runits, period, Tunits, period_min
      write(*,268) r0overre
      write(*,250)
      if(dosort) write(*,270)
      write(*,272) filename
      write(*,273) imon, iday, iyr,  ihr, iminute, isecond
  130 format(A1)
  160 format(A24)
  250 format(1H ,'           ')
  255 format(1h ,' BASIS VECTORS USED ARE PHI, Q, P, Q**2, QP, P**2, ETC
     &. ')
  258 format(1H ,' KEPLER PROBLEM in a PLANE, DEVIATIONS FROM CIRCULAR O
     &RBIT Q.')
  259 format(1H ,'OBTAINING EIGENVALUES FOR THE LIOUVILLE GENERATED MATR
```

```
     $IX FOR KEPLER PROBLEM',/,5x,' IN A PLANE W/ RADIAL DEVIATIONS FROM
     $ A CIRCULAR EQUILIBRIUM',/,5x,'ORBIT.')
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 format(1H ,' Sat XM is:',D11.4,' Kg    GM is:',D14.6,1x,A8)
  262 format(1H ,' Rinit_radial is:',D12.6,1x,A2
     &   ,/,10x,' (ABOVE USED IN CALC INITIAL ANGULAR MOMENTUM VECTOR)')
  263 format(1H ,' Vinit_radial is:',D12.6,1x,A6,1x,' Vinit_trans is:',D
     &12.6,1x,A6,/,10x,'(ABOVE USED IN CALC INITIAL ANGULAR MOMENTUM VEC
     &TOR)')
  265 format(1H ,' INITIAL ANGULAR MOMENTUM VECTOR IS:',3(D12.6,2x) )
  266 format(1H ,' MAGNITUDE SQ OF INITIAL ANG MOM VECTOR is:'
     &,D16.8,' (',A10,')**2',/,10x,' ANG MOM MAGNITUDE IS: ',D16.8,1x,
     & A10)
  267 format(1H ,' EQUILIBRIUM RADIUS RO IS:',D12.6,1x,A2,' PERIOD IS: '
     &,D12.6,1x,A3,/,10x,' PERIOD in MINUTES is: ',D12.6)
  268 format(1H ,' Equilbrium radius in EARTH RADII is: ',D12.6)
  270 format(1H ,' EIGENVALUES PROCESSED TO RUN FROM LARGEST TO SMALLEST
     - WITHIN EACH',/,'   SUB-PARTITION OF THE L MATRIX.')
  272 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  273 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
c-->     Define the A matrix for 10x10 truncation......
      do 1000 j = 1, nmax
        do 1000 i = 1, nmax
          a(i,j) = 0.0D+00
 1000 continue
c-->    Define L  matrix for up to n=10 truncation....
      a(1,2) = -2.0D+00*alpha*xlphi

      a(2,3) = alpha
      a(3,2) = -3.0D+00*alpha*xlphisq + 2.0D+00*beta

      a(1,4) =  3.0D+00*alpha*xlphi
      a(3,4) = 3.0D+00*(2.0D+00*alpha*xlphisq - beta)
      a(4,5) =  (2.0D+00)*alpha
      a(5,2) =  alpha*xlphisq - beta
      a(5,4) =  a(3,2)
      a(5,6) =  alpha
      a(6,3) = (2.0D+00)*a(5,2)
      a(6,5) = (2.0D+00)*a(3,2)

      a(1,7) = -4.0D+00*alpha*xlphi
      a(5,7) =  a(3,4)
      a(6,8) = (2.0D+00)*a(3,4)
      a(7,8) = (3.0D+00)*alpha
      a(8,4) =  a(5,2)
      a(8,7) =  a(3,2)
      a(8,9) = (2.0D+00)*alpha
```

```
      a(9,5) = (2.0D+00)*a(5,2)
      a(9,8) =  2.0D+00*a(3,2)
      a(9,10) =  alpha
      a(10,6)  =  3.0D+00*a(5,2)
      a(10,9) =  3.0D+00*a(3,2)
c-->    Check for exceedingly small residual elements....
c         Zero all elements whose magnitude is less than 10 times the
c         machine epsilon (machep)
      do 1111 j = 1, nmax
        do 1112 i = 1, nmax
          if( (dabs(a(i,j))) .le. 1.0D-12 ) a(i,j) = 0.0D+00
 1112   continue
 1111 continue
      if(idoprint) then
c-->      Write out A matrix....
        write(*,'('' Portion of A matrix used is:  '')' )
        write(6,'('' Portion of A matrix used is:  '')' )
        do 1180 k = 1, n
          write(*,'('' row '',i3)') k
          write(*,'('' '',5(D14.7,1x),/)' ) (a(k,j), j = 1,n)
 1180   continue
        do 1190 k = 1, n
          write(6,'('' row '',i3)') k
          write(6,'('' '',5(D14.7,1x),/)' ) (a(k,j), j = 1,n)
 1190   continue
      endif
      write(*,'('' '',71(1H*))' )
      write(5,'('' '',71(1H*))' )
      write(6,'('' '',71(1H*))' )
      write(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
      write(6,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
c     ***********************************************************************
c       Proceed to get eigenvalues of the A matrix.
c         matz = 0 get eigenvalues only; matz = 1 get eigenvectors also
      matz = 0
      nm = nmax
c-->    Use the driver routine RG to get eigenvalues.
c     call rg(nm,n,a,wr,wi,0,z,iv1,fv1,ierr)
c-->    Use the driver routine RUHESS to get eigenvalues.
      call ruhess(nm,n,a,wr,wi,0,z,iv1,fv1,ierr)
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c-->     Unpack eigenvector values from the NxN Z array, per
c          instructions and code from Section 2.3-6 of the Eispack Guide
c          pages 88-89
c     do 150 k = 1, n
c       if(wi(k) .ne. 0.0) go to 110
c       do 100 j = 1, n
c         zr(j,k) = z(j,k)
```

```
c        zi(j,k) = 0.0
c 100   continue
c       go to 150
c 110   if(wi(k) .lt. 0.0) go to 130
c       do 120 j = 1, n
c         zr(j,k) = z(j,k)
c         zi(j,k) = z(j,k+1)
c 120   continue
c       go to 150
c 130   do 140 j = 1, n
c         zr(j,k) = zr(j,k-1)
c         zi(j,k) = -zi(j,k-1)
c 140   continue
c 150 continue
      if(ierr .ne. 0) then
        if(ierr .lt. 0) then
          write(*,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(6,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(5,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(*,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(6,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(5,'('' Terminating program---IERR is: '',i3)' ) ierr
          stop 00001
        elseif(ierr .gt. 0) then
          write(*,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(6,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(5,'('' At least one eigenvalue failed to converge in 30
     & iterations. Failure occurred for eigenvalue having index, '',i3
     &)' ) ierr
          write(*,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(6,'('' Terminating program---IERR is: '',i3)' ) ierr
          write(5,'('' Terminating program---IERR is: '',i3)' ) ierr
          stop 00002
        else
        endif
      else
c-->     IERR is zero, indicating no convergence problems--print results
c          Limit real and imaginary parts to greater than 1.0D-12
```

```
      do 1331 ii = 1, n
        if( dabs(wr(ii)) .lt. 1.0D-12 ) wr(ii) = 0.0D+00
        if( dabs(wi(ii)) .lt. 1.0D-12 ) wi(ii) = 0.0D+00
 1331 continue
      write(*,'('' '')' )
      write(5,'('' '')' )
      write(6,'('' '')' )
c-->  Perform sort, if requested....
c     if(dosort) then
c-->    Sort eigenvalue magnitudes on all partitions except the first
c         Pack into the various partitons the eigenvalues in
c           increasing order
c       do 2500 j = 1, npartshn -1
c         lswap = .FALSE.
c         numinpart = kpartshn(j+1) - kpartshn(j)
c         if( numpartshn .gt. 2) then
c           do 2800 k = 1, numinpart - 2
c             if( .not. (dabs(wr(kpartshn(j) + k)) .gt. dabs(wr(
c    &        kpartshn(j) + k + 2) ) .or. dabs(wi(kpartshn(j) + k))
c    &        .gt. dabs(wi(kpartshn(j) + k + 2)) ) .and. .not. lswap)
c    &        then
c               tempr = wr(kpartshn(j) + k)
c               wr(kpartshn(j) + k) = wr(kpartshn(j) + k + 2)
c               wr(kpartshn(j) + k + 2) =  tempr
c               wr(kpartshn(j) + k + 3) = -tempr
c               tempi = wi(kpartshn(j) + k)
c               wi(kpartshn(j) + k) = wi(kpartshn(j) + k + 2)
c               wi(kpartshn(j) + k + 2) =  tempi
c               wi(kpartshn(j) + k + 3) = -tempi
c               lswap = .TRUE.
c             endif
c2800       continue
c           endif
c2500     continue
c       endif
      write(5,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
      do 300 i = 1, n
        if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &    (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) )
     &    write(*,'('' '')')
        if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &    (i.eq.(kpartshn(3)+1)) .or. (i.eq.(kpartshn(4)+1)) )
     &    write(6,'('' '')')
        write(*,'('' i = '',i3,'' EIGENVALUE: '',D16.10,1x,D16.10,''
     & (I) '')' ) i, wr(i), wi(i)
        write(6,'('' i = '',i3,'' EIGENVALUE: '',D16.10,1x,D16.10,''
     & (I) '')' ) i, wr(i), wi(i)
c-->      Write eigenvalues to output file for program use ....
```

```
         write(5,'(D16.10,2x,D16.10)') wr(i), wi(i)
  300   continue
        endif
        write(*,'('' '',71(1H*))' )
        write(6,'('' '',71(1H*))' )
        close(6)
        close(5)
        end
c******************** END of PROGRAM EVKEPL10 ************************
$DEBUG
      subroutine crosspr(a,b,c)
c*************************************************************************
c******* This subroutine, provided with the vectors a and b, which ****
c******* have real components, performs double precision arithmetic ***
c******* to obtain the cross product of the vectors a and b.  The cross
c******* product operation is defined in the standard manner.  The ****
c******* result is stored in the vector c. All vectors have 3 component
c*************************************************************************
      implicit real*8 (a-h,o-z)
      dimension a(3), b(3), c(3)
      c(1)= a(2)*b(3) - b(2)*a(3)
      c(2)= a(3)*b(1) - b(3)*a(1)
      c(3)= a(1)*b(2) - b(1)*a(2)
      return
      end
c******************** END of SUBROUTINE CROSSPR ***********************
$DEBUG
      double precision function dot(a,b)
c*************************************************************************
c******* This function, provided with the vectors a and b, which have**
c******* real components, performs double precision arithmetic to *****
c******* obtain the dot product of the vectors a and b.  The dot ******
c******* product operation in the standard manner. ********************
c******* All vectors are assumed to have 3 or less components. *******
c*************************************************************************
      implicit real*8 (a-h,o-z)
      dimension a(3), b(3)
      dot = a(1)*b(1) + a(2)*b(2) + a(3)*b(3)
      return
      end
c************ END of DOUBLE PRECISION FUNCTION DOT  ******************
$DEBUG
c************SUBROUTINE RUHESS (EISPACK Guide Modified RG) ***************
c************ Source listing taken from B.T. SMITH et. al. 1979 ********
c************    Matrix Eigensystem Routines - EISPACK guide 2nd Ed ****
c*************************************************************************
c
c     ******* MODIFIED 10/29/87 to only call balanc and hqr
```

```
c     *******  Since the input matrix is upper Hessenberg      TLH
c
      subroutine ruhess(nm,n,a,wr,wi,matz,z,iv1,fv1,ierr)
      implicit real*8 (a-h,o-z)
      integer n,nm,is1,is2,ierr,matz
      real*8 a(nm,n),wr(n),wi(n),z(nm,n),fv1(n)
      integer iv1(n)
      if(n .le. nm) go to 10
      ierr = 10 * n
      go to 50
   10 call balanc(nm,n,a,is1,is2,fv1)
c     call elmhes(nm,n,is1,is2,a,iv1)
      if(matz .ne. 0) go to 20
c     ****************  FIND EIGENVALUES ONLY  **************************
      call hqr(nm,n,is1,is2,a,wr,wi,ierr)
      go to 50
c     ****************  FIND BOTH EIGENVALUES AND EIGENVECTORS  *********
   20 call eltran(nm,n,is1,is2,a,iv1,z)
      call hqr2(nm,n,is1,is2,a,wr,wi,z,ierr)
      if(ierr .ne. 0) go to 50
      call balbak(nm,n,is1,is2,fv1,n,z)
   50 return
      end
c*******************  END OF SUBROUTINE RUHESS ************************
```

LISTINGS FOR SUBROUTINES BALANC, HQR, ELTRA

AND BALBAK ARE CONTAINED IN

THE EISPACK GUIDE (SMITH et al, 1976)

APPENDIX H

FORTRAN LISTING FOR PUTZER ALGORITHM FOR 6x6 TRUNCATION
OF L MATRIX FOR KEPLER PROBLEM

```
$DEBUG
      program putkep6
c*************************************************************************
c*******  This program performs the Putzer algorithm for the  6x6  trunc
c*******  of the L matrix which is Upper (Hessenberg) found from the ***
c*******  action of the Liouville operator upon a polynomial basis in **
c*******  phi, and powers of q and p.  Eigenvalues are found by using **
c*******  a modified version of the driver subroutine RG from the ******
c*******  EISPACK guide. ***********************************************
c*******  Eigenvalues are read in from the file EVKEP10.PGM  6/MAY/88 **
c*******  The L matrix used must be a even dimensioned matrix for the **
c*******  EISPACK routines to return the proper eigenvalues.  Because **
c*******  of this, the final expression for phi(t) must include alpha **
c*******  times xlphi (which is omega) times the time.  This is the ****
c*******  only contribution that is left out when chopping the initial *
c*******  11x11 L matrix down to a 10x10 by deleting the first row and *
c*******  column of the 11x11 L matrix and the basis vector of 1. ******
c*******  Routines BUILDP and GETRK calculate pieces required by the ***
c*******  Putzer Algorithm for calculating e**(Lt). ********************
c*******  If no external radial force is applied the matrix reduces  ***
c*******  to an upper Hessenberg matrix. The matrix is partitioned to **
c*******  to obtain its general structure with respect to the polynomial
c*******  Using the eigenvalue information, the scalar functions R(t) **
c*******  are produced by the numerical integration using the Runge- ***
c*******  Kutta Gill method.  A basis consisting of phi, and polynomials
c*******  consisting of q, the radial deviation from an equilibrium  ***
c*******  circular orbit, and p, the corresponding conjugate momenta.***
c*******  The Hamiltonian used is that for the Kepler problem in a   ***
c*******  plane.  Two sets of units are provided in which the problem***
c*******  can be worked.  They are the physical units provided by the **
c*******  MKS system, or the canonical units, determined in such a way *
c*******  that GM = 1. *************************************************
c********   The program prompts the user for a filename (DOS) to     ***
c********     which it will write the resultant data.                ***
c********   The Hamiltonian is as follows: ***************************
c********                                                            ***
c********   In MKS units...                                          ***
c********     H =  (Pr)*(Pr)/(2m*r0*r0) + ((lphi)**2)/(2*m*r0*r0*    ***
c********           (1 + Q)**2) - GMm/(r0*(1+Q))                     ***
c********                                                            ***
c********   In CANONICAL units....                                   ***
```

```
c********     H =  (Pr)*(Pr)/(2*r0*r0) + ((lphi)**2)/(2*r0*r0*      ***
c********          (1 + Q)**2) -   1/(r0*(1+Q))                     ***
c********                                                           ***
c********  The user is prompted for his choice of units to work in. ***
c********                                                           ***
c*************************************************************************
c*******  Units are in the MKS system. Kg, Km, sec OR a CANONICAL UNIT *
c*******  system Kg, Radius of earth in Km, CTU (Canonical Time Unit) **
c*******  = 806.5 sec = 13.44 min, at the choice of the user. **********
c*************************************************************************
c*******  Cylindrical coordinate system is assumed.  r, phi, z ********
c*************************************************************************
c*************************************************************************
      parameter(nmax=6,npartshn=3)
      implicit real*8 (a-h,o-z)
      real*8 a(nmax,nmax), wr(nmax), wi(nmax)
      real*8 rinit(3), vinit(3), amominit(3), c(3)
      dimension linefo(6)
      complex*16 ca(nmax,nmax), ceigval(nmax), cident(nmax,nmax)
      complex*16 pfirst(nmax,nmax), psecond(nmax,nmax), temp(3,nmax)
      complex*16 pthird(nmax,nmax)
      complex*16 expmatt(3,nmax)
      complex*16 u(nmax), u0(nmax), r_rkgill(nmax), r_last(nmax)
      integer n, nn, matz, iv1(nmax), ierr, kpartshn(npartshn)
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      character*24 filename
      character*24 ev_file
      character*12 linefo
      character*1  idoprint
      character*1  manswer
      character*1  iread_ev_file
      character*1  istepans
      character*1 i_pick_units
      character*8 GMunits
      character*2 Runits
      character*3 Tunits
      character*6 Vunits
      character*9 Punits
      character*10 AMunits
      logical canonical /.FALSE./
      logical ichgstep /.FALSE./
      logical doprint /.FALSE./
      logical docheckr /.FALSE./
      logical lswap /.FALSE./
      logical initialze /.TRUE./
      logical no_default_ev /.FALSE./
      common /matrices/ ca, cident, n
      common /pmatrix/  pfirst, psecond, pthird
```

```
      data ev_file /'C:\evkep10.pgm         '/
      data h/.10D+00/
c-->     Physical constant data....
      data convmkm/1.0D-03/
      data pi/3.1415927D+00/
      data G/6.67D-17/, earth_mas/5.98D+24/, r_earth/6.378165D+03/
      data ctu_per_sec/1.2398521D-03/
c-->  Call GETDAT and GETTIM (obtain system date and time)
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
c-->     Define partioning structure of the matrix...
c-->  Ending partition indices are given by kpartshn(j) for
c                                                 j = 1,...,npartshn
c     Beginning partition indices are given by kpartshn(l)+1 for
c                                              l = 1,...,npartshn-1
      do 450 j = 1, npartshn
        kpartshn(j) = ( (j*(j+1))/2 )
  450 continue
      write(*,'('' Input the dimension(order) .le.'',I3,'', of the A mat
     &rix.'')' ) kpartshn(npartshn)
      read(*,*) n
      write(*,'('' Do you want the matrix printed out? y or n '')')
      read(*,130) manswer
      if((manswer .eq. 'y' ) .or. (manswer .eq. 'Y')) doprint = .TRUE.
c-->       now get filename to use in writing the output...
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(6,file=filename,status='new')
      write(*,'('' Do you wish to enter the path and filename of the fil
     &e containing the computed eigenvalues? (Y/N)'')')
      read(*,'(A1)') iread_ev_file
      if((iread_ev_file.eq.'Y').or.(iread_ev_file.eq.'y')) no_default_ev
     & = .TRUE.
      if(no_default_ev) then
        write(*,'('' Enter eigenvalue data PATH and FILENAME'')')
        read(*,'(A24)') ev_file
      endif
      open(7,file=ev_file,status='unknown')
      rewind 7
      write(*,'('' Input the value of m in Kg:'',/,
     &'' Use the format:'',/,
     &'' D11.4'' ,/,
     &'' Sd.ddddD+XX'' )')
      read(*,1229) xm
 1229 format(1x,D11.4)
      write(*,'('' Default coord system units is the MKS system.'')')
      write(*,'('' Do you wish to use CANONICAL system of units? Enter
```

```
     &Y or N'')')
      read(*,'(A1)') i_pick_units
      if((i_pick_units .eq. 'Y') .or. (i_pick_units .eq. 'y')) canonical
     & =  .TRUE.
c-->    Input initial components for position vector....
      write(*,'('' Input initial values RINITIAL(radial) in Km: '',/,
     &'' Use the format:'',/,
     &'' D14.7'' ,/,
     &'' Sd.ddddddd+XX'' )')
      read(*,1231) rinit_radial
 1231 format(1x,D14.7)
 1232 format(1x,D14.7)
      write(*,1232) rinit_radial
      if( .not. (canonical) ) then
        GM = G*earth_mas*convmkm
        rinit(1) = rinit_radial
        rinit(2) = 0.0D+00
        rinit(3) = 0.0D+00
      else
        GM = 1.0D+00
        xm = 1.0D+00
        rinit(1) = rinit_radial/r_earth
        rinit(2) = 0.0D+00
        rinit(3) = 0.0D+00
      endif
c-->    Input initial components for momentum vector....
      write(*,'('' Input initial velocity values VINITIAL(radial), and V
     &INITIAL(transverse): in Km/sec '',/,
     &'' Use the format:'',/,
     &'' D14.7,1X,D14.7'' ,/,
     &'' Sd.ddddddd+XXbSd.ddddddd+YY'' )')
      read(*,1233) vinit_radial, vinit_trans
      write(*,1234) vinit_radial, vinit_trans
 1233 format(1x,D14.7,1x,D14.7)
 1234 format(1x,D14.7,1x,D14.7)
      if( .not. (canonical) ) then
        vinit(1) = vinit_radial
        vinit(2) = vinit_trans
        vinit(3) = 0.0D+00
      else
        vinit(1) = (vinit_radial/r_earth)/ctu_per_sec
        vinit(2) = (vinit_trans/r_earth)/ctu_per_sec
        vinit(3) = 0.0D+00
      endif
c-->    Input initial values for phi, q, and p.....
      write(*,'('' Input initial values of PHI, radial deviation Q, and
     &P: in rad, unitless, & Kg-Km2/sec'',/,
     &'' Use the format:'',/,
```

```
     &'' D14.7,1X,D14.7,1X,D14.7'' ,/,
     &'' Sd.dddddddD+XXbSd.dddddddD+YYbSd.dddddddD+ZZ'' )')
      read(*,1235) phizero, qzero, pzero
      write(*,1236) phizero, qzero, pzero
 1235 format(1x,D14.7,1x,D14.7,1x,D14.7)
 1236 format(1x,D14.7,1x,D14.7,1x,D14.7)
      if(canonical) then
        pzero =  (pzero/(r_earth*r_earth))/ctu_per_sec
      endif
c-->       Calculation of initial angular momentum....
      call crosspr(rinit,vinit,c)
      amominit(1) = xm*c(1)
      amominit(2) = xm*c(2)
      amominit(3) = xm*c(3)
c-->       Calculation of magnitude of initial angular momentum....
      xlphisq = dot(amominit,amominit)
      xlphi = dsqrt(xlphisq)
c-->       Calculation of circular orbit radius....
      if( .not. (canonical) ) then
        r0 = xlphisq/(GM*xm*xm)
        r0overre = r0/r_earth
      else
        r0 = xlphisq
        r0overre = r0
      endif
        aa  = r0
c-->       Calculation of initial period using Kepler's Third Law
      if( .not. (canonical) ) then
        period = (2.0D+00*pi*aa*dsqrt(aa))/dsqrt(GM)
        period_min = period/60.0D+00
      else
        period = 2.0D+00*pi*aa*dsqrt(aa)
        period_min = period*(1.0D+00/ctu_per_sec)*(1.0D+00/60.0D+00)
      endif
      write(*,'('' Input the init time, no of orbits, and no.of steps.''
     &,/,''  Use the format:'',/,
     &'' D10.4,1X,D10.4,1X,I5'' ,/,
     &'' Sd.dddD+XXbSd.dddD+YYb54321'' )')
      read(*,1245) tbegin, znorbits, nsteps
 1245 format(1x,D10.4,1x,D10.4,1x,I5)
c-->       Calculate the increment to advance the time with
      tend = znorbits*period
      deltat = (tend - tbegin)/dble(nsteps)
      t0 = tbegin
      write(*,'(''  TEND is:'',D12.6,1x,A3)') tend, Tunits
      write(*,'('' Input the time at which to check calculations.'',/,
     &''  Use the format:'',/,
     &'' D14.7'' ,/,
```

```
     &'' Sd.ddddddD+XX'' )')
      read(*,1433) tchkcalc
 1433 format(1x,D14.7)
      if(tchkcalc .gt. 0.0D+00) docheckr = .TRUE.
      write(*,'('' Size of DELTAT is:'',D12.6,1x,A3)') deltat, Tunits
      write(*,'('' Want STEPSIZE .LE. DELTAT '')')
      write(*,'('' Default value of integration STEPSIZE is:'',D10.4)')
     & h
      write(*,'('' Do you wish to change it? Enter (Y/N)'')')
      read(*,'(A1)') istepans
      if((istepans .eq. 'y') .or. (istepans .eq. 'y')) ichgstep = .TRUE.
      if(ichgstep) then
        write(*,'(1x,''Enter the value (>= 1) to divide DELTAT by to pro
     &duce the value for STEPSIZE, use format'')')
        write(*,'(1x,''Sd.ddddddD+XX'' )')
        read(*,'(D14.7)') hfactor
        write(*,'(D14.7)') hfactor
        h = deltat/hfactor
        write(*,'('' RKG STEPSIZE is: '',D14.7)') h
      endif
c-->    Calculate the constants used in the matrix of the L matrix.....
      if( .not. (canonical) ) then
        alpha = 1.0D+00/(xm*r0*r0)
        alphasq = alpha*alpha
        beta  = GM*xm/r0
      else
        alpha = 1.0D+00/(r0*r0)
        alphasq = alpha*alpha
        beta = 1.0D+00/r0
      endif
c       ....write data to output file for records ...
      if( .not. (canonical) ) then
        GMunits = 'Km3/sec2'
        Runits  = 'Km'
        Vunits  = 'Km/sec'
        Tunits  = 'sec'
        Punits  = 'Kg-Km/sec'
        AMunits = 'Kg-Km2/sec'
      else
        GMunits = 'Re3/CTU2'
        Runits  = 'Re'
        Vunits  = 'Re/CTU'
        Tunits  = 'CTU'
        Punits  = 'Kg-Re/CTU'
        AMunits = 'Kg-Re2/CTU'
      endif
      write(6,259)
      write(6,258)
```

```
      write(6,255)
      write(6,250)
      write(6,260)
      write(6,261) xm, GM, GMunits
      write(6,262) rinit(1), Runits
      write(6,263) vinit(1), Vunits, vinit(2), Vunits
      write(6,250)
      write(6,264) phizero, qzero, pzero
      write(6,265) (amominit(i), i = 1,3)
      write(6,266) xlphisq, AMunits, xlphi, AMunits
      write(6,267) r0, Runits, period, Tunits, period_min, znorbits
      write(6,268) r0overre
      write(6,250)
      write(6,270) nsteps
      write(6,271) tbegin, tend, Tunits, deltat, Tunits, h, Tunits
      write(6,272) filename
      write(6,273) imon, iday, iyr,  ihr, iminute, isecond
C         ....write data to screen for verification ...
      write(*,259)
      write(*,258)
      write(*,255)
      write(*,250)
      write(*,260)
      write(*,261) xm, GM, GMunits
      write(*,262) rinit(1), Runits
      write(*,263) vinit(1), Vunits, vinit(2), Vunits
      write(*,250)
      write(*,264) phizero, qzero, pzero
      write(*,265) (amominit(i), i = 1,3)
      write(*,266) xlphisq, AMunits, xlphi, AMunits
      write(*,267) r0, Runits, period, Tunits, period_min, znorbits
      write(*,268) r0overre
      write(*,250)
      write(*,270) nsteps
      write(*,271) tbegin, tend, Tunits, deltat, Tunits, h, Tunits
      write(*,272) filename
      write(*,273) imon, iday, iyr,  ihr, iminute, isecond
  130 format(A1)
  160 format(A24)
  250 format(1H ,'           ')
  255 format(1h ,' BASIS VECTORS USED ARE PHI, Q, P, Q**2, QP, P**2 =6x6
     &. ')
  258 format(1H ,' KEPLER PROBLEM in a PLANE, DEVIATIONS FROM CIRCULAR O
     &RBIT Q.')
  259 format(1H ,'PERFORMING PUTZER ALGORITHM TO CALCULATE EXP(Lt) FOR T
     $HE FOR KEPLER PROBLEM',/,5x,' IN A PLANE W/ RADIAL DEVIATIONS FROM
     $ A CIRCULAR EQUILIBRIUM',/,5x,'ORBIT.')
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
```

```
  261 format(1H ,' Sat XM is:',D11.4,' Kg    GM is:',D14.6,1x,A8)
  262 format(1H ,' Rinit_radial is:',D12.6,1x,A2
     &   ,/,10x,' (ABOVE USED IN CALC INITIAL ANGULAR MOMENTUM VECTOR)')
  263 format(1H ,' Vinit_radial is:',D12.6,1x,A6,1x,' Vinit_trans is:',D
     &12.6,1x,A6,/,10x,'(ABOVE USED IN CALC INITIAL ANGULAR MOMENTUM VEC
     &TOR)')
  264 format(1H ,' PHIZERO IS:',D12.6,' Rad  QZERO IS:',D12.6,' PZERO IS
     &:',D12.6,/,10x,' (ABOVE USED IN BUILDING INITIAL CONDITION VECTOR)
     &')
  265 format(1H ,' INITIAL ANGULAR MOMENTUM VECTOR IS:',3(D12.6,2x) )
  266 format(1H ,' MAGNITUDE SQ OF INITIAL ANG MOM VECTOR is:'
     &,D16.8,' (',A10,')**2',/,10x,' ANG MOM MAGNITUDE IS: ',D16.8,1x,
     & A10)
  267 format(1H ,' EQUILIBRIUM RADIUS RO IS:',D12.6,1x,A2,' PERIOD IS: '
     &,D12.6,1x,A3,/,10x,' PERIOD in MINUTES is: ',D12.6,1x,'No of ORBIT
     &S is: ',D12.6)
  268 format(1H ,' Equilbrium radius in EARTH RADII is: ',D12.6)
  270 format(1H ,'INTEGRATION LIMITS FOLLOW;        NSTEPS IS ',I6)
  271 format(1H ,' TBEGIN = ',D12.6,' ; TEND = ',D12.6,1x,A3,' DELTAT =
     -',D14.7,1x,A3,/,5x,' and RKG STEPSIZE is: ',D14.7,1x,A3)
  272 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  273 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)

c-->    Define the A matrix for  6x6  truncation......
      do 1000 j = 1, nmax
        do 1000 i = 1, nmax
          a(i,j) = 0.0D+00
 1000 continue
c-->   Define L  matrix for up to n=6  truncation....
      a(1,2) = -2.0D+00*alpha*xlphi

      a(2,3) = alpha
      a(3,2) = -3.0D+00*alpha*xlphisq + 2.0D+00*beta

      a(1,4) =  3.0D+00*alpha*xlphi
      a(3,4) = 3.0D+00*(2.0D+00*alpha*xlphisq - beta)
      a(4,5) =  (2.0D+00)*alpha
      a(5,2) =  alpha*xlphisq - beta
      a(5,4) =  a(3,2)
      a(5,6) =  alpha
      a(6,3) = (2.0D+00)*a(5,2)
      a(6,5) = (2.0D+00)*a(3,2)

c     a(1,7) = -4.0D+00*alpha*xlphi
c     a(5,7) =  a(3,4)
c     a(6,8) = (2.0D+00)*a(3,4)
c     a(7,8) = (3.0D+00)*alpha
```

```
c     a(8,4) =  a(5,2)
c     a(8,7) =  a(3,2)
c     a(8,9) = (2.0D+00)*alpha
c     a(9,5) = (2.0D+00)*a(5,2)
c     a(9,8) =  2.0D+00*a(3,2)
c     a(9,10) =  alpha
c     a(10,6)  =  3.0D+00*a(5,2)
c     a(10,9) =  3.0D+00*a(3,2)
c-->    Check for exceedingly small residual elements....
c         Zero all elements whose magnitude is less than 1.0D-12
      do 2111 j = 1, nmax
        do 2112 i = 1, nmax
          if( (dabs(a(i,j))) .le. 1.0D-12 ) a(i,j) = 0.0D+00
 2112   continue
 2111 continue
      if(doprint) then
c-->      Write out A matrix....
        write(*,'('' Portion of A matrix used is:  '')' )
        write(6,'('' Portion of A matrix used is:  '')' )
        do 1180 k = 1, n
          write(*,'('' row '',i3)') k
          write(*,'('' '',5(D14.7,1x),/)' ) (a(k,j), j = 1,n)
 1180   continue
        do 1190 k = 1, n
          write(6,'('' row '',i3)') k
          write(6,'('' '',5(D14.7,1x),/)' ) (a(k,j), j = 1,n)
 1190   continue
      endif
      write(*,'('' '',71(1H*))' )
      write(6,'('' '',71(1H*))' )
c-->    convert matrix A into matrix CA, its complex form
      do 1230 jj = 1, n
        do 1240 ii = 1, n
          ca(ii,jj) = dcmplx( a(ii,jj), 0.0D+00 )
 1240   continue
 1230 continue
c-->    Form the vector of complex initial valued basis parameters
      u0(1) = dcmplx(phizero, 0.0D+00 )

      u0(2) = dcmplx(qzero, 0.0D+00 )
      u0(3) = dcmplx(pzero, 0.0D+00 )

      u0(4) = dcmplx(qzero*qzero, 0.0D+00 )
      u0(5) = dcmplx(qzero*pzero, 0.0D+00 )
      u0(6) = dcmplx(pzero*pzero, 0.0D+00 )
c-->    Build complex identity matrix of order n
      do 1250 jj = 1, n
        do 1260 ii = 1, n
```

```
        if(jj .eq. ii) then
          cident(ii,ii) = dcmplx(1.0D+00, 0.0D+00)
        else
          cident(ii,jj) = dcmplx(0.0D+00, 0.0D+00)
        endif
 1260   continue
 1250 continue
c--> ##################################################################
c      Read in the eigenvalues from the file EVKEP10.PGM.(default) or
c        from previously entered file.
c        First read past header containing info on physical constants
c        used to obtain the set of eigenvalues, and file creation info.
c
c-->     Read file until the header is passed.  The header is delimited
c-->     by a line that consists of the character string,
c-->     +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/
c-->     as a delimiter.
c-->     read to get past header on file known as 'ev_file'.
 1111 read(7,'(6(A12))',end=215) linefo
      if((linefo(2) .eq. '+/+/+/+/+/+/') .or. (linefo(2) .eq.
     $ '/+/+/+/+/+/+') ) then
        go to 500
      else
        go to 1111
      endif
c-->    re-set linefo.....
  500 do 335 ik = 1,6
        linefo(ik) = '            '
  335 continue
c--> we will open the file and read in the eigenvalues for the current
c-->     truncation of the L matrix.
      do 10 i = 1, n
        read(7,'(D16.10,2x,D16.10)',end=225) wr(i), wi(i)
   10   continue
c     ##### Form complex eigenvalue vector ......
        do 300 i = 1, n
          ceigval(i) = dcmplx( wr(i),wi(i) )
  300   continue
        if(doprint) then
          write(*,'('' Complex eigenvalue vector follows: '')')
          write(6,'('' Complex eigenvalue vector follows: '')')
          do 325 i = 1, n
            if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &        (i.eq.(kpartshn(3)+1)) )
     &        write(*,'('' '')')
            if( (i.eq.(kpartshn(1)+1)) .or. (i.eq.(kpartshn(2)+1)) .or.
     &        (i.eq.(kpartshn(3)+1)) )
     &      write(6,'('' '')')
```

```
          write(*,'('' i = '',i3,'' EIGENVALUE: '',D14.8,1x,D14.8,''(
     &I)'')' ) i, wr(i), wi(i)
          write(6,'('' i = '',i3,'' EIGENVALUE: '',D14.8,1x,D14.8,''(
     &I)'')' ) i, wr(i), wi(i)
  325     continue
        endif
c         Call routine BuildP to build two storage areas.  The first
c           PFIRST contains the first row of the P matrix having indice
c           INDEX, e.g. PFIRST(INDEX,row). The second, PSECOND, contains
c           the second row of the P matrix having indice INDEX, e.g.
c           PSECOND(INDEX,row).  The third, PTHIRD, contains the third
c           row of the P matrix having indice INDEX, e.g. PTHIRD(INDEX,
c           row).
      call buildp6(ceigval)
c-->      Write out the PFIRST, PSECOND & PTHIRD matrices, if requested.
      if(doprint) then
        write(*,'('' PFIRST. PSECOND, PTHIRD Matrices follow:   '')' )
        write(6,'('' PFIRST. PSECOND, PTHIRD Matrices follow:   '')' )
        do 3170 k = 1, n
          write(*,'('' P matrix index is '',i3)') k
            write(*,'('' row 1 '')')
            write(*,'('' '',3(2(G11.4,1x),1x),/)')(pfirst(k,j), j = 1,n)
            write(*,'('' row 2 '')')
            write(*,'('' '',3(2(G11.4,1x),1x),/)' )(psecond(k,j),j= 1,n)
            write(*,'('' row 3 '')')
            write(*,'('' '',3(2(G11.4,1x),1x),/)' )(pthird(k,j),j= 1,n)
            write(*,'(1x,70(1H-))')
 3170   continue
        do 4170 k = 1, n
          write(6,'('' P matrix index is '',i3)') k
            write(6,'('' row 1 '')')
            write(6,'('' '',3(2(G11.4,1x),1x),/)')(pfirst(k,j), j = 1,n)
            write(6,'('' row 2 '')')
            write(6,'('' '',3(2(G11.4,1x),1x),/)' )(psecond(k,j),j= 1,n)
            write(6,'('' row 3 '')')
            write(6,'('' '',3(2(G11.4,1x),1x),/)' )(pthird(k,j),j= 1,n)
            write(6,'(1x,70(1H-))')
 4170   continue
        write(*,'('' '',71(1H-))' )
        write(6,'('' '',71(1H-))' )
      endif
c     ********** BEGIN ITERATING IN TIME **********************
      t = t0
         write(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+
     &/'')')
         write(6,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+
     &/'')')
      do 40 k = 1, nsteps + 1
```

```
        tlast = t
        if((t .eq. 0.0D+00) .and. (initialze)) then
          r_rkgill(1) = dcmplx(1.0D+00, 0.0D+00)
          do 8200 kl = 2, n
            r_rkgill(kl) = dcmplx(0.0D+00, 0.0D+00)
 8200     continue
          initialze = .FALSE.
        else
c       Define last values of the r_rkgill(n) vector for use as the
c       initial condition vector for the RKGILL integration.
          do 8300 kn = 1, n
            r_last(kn) = r_rkgill(kn)
 8300     continue
c-->    Get the entire set of polynomial functions, each evaluated at the
c         integrated time T, given that the set of polynomials have the
c         initial conditions given by the values contained within the
c         the vector r_last(n).....
c
c       Pass DT to determine the number of times to execute the Runge-
c         Kutta-Gill integrator routine...
          call getrk6(h,n,ceigval,deltat,r_last,r_rkgill,tlast,t)
c-----------------------------------------------------------------------
        endif
c       Initialize array to hold results of this particular time.
c           (iteration of deltat)
        do 3050 ii = 1, 3
          do 3060 jj = 1, n
            expmatt(ii,jj) = dcmplx(0.0D+00,0.0D+00)
 3060     continue
 3050   continue
        if((docheckr) .and. (dabs(t - tchkcalc) .le. .008D+00)) then
          do 8400 kn = 1, n
            write(*,'('' For r sub '',i3,'', r is: '',D14.8,2x,D14.8)')
     &        kn, r_rkgill(kn)
 8400     continue
          do 8500 kn = 1, n
            write(6,'('' For r sub '',i3,'', r is: '',D14.8,2x,D14.8)')
     &        kn, r_rkgill(kn)
 8500     continue
          if(t .eq. 0.0D+00 .and. (docheckr)) then
          write(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+
     &/'')')
          write(6,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+
     &/'')')
          endif
        endif
c-->      Calculate the terms in the Putzer sum, accumulating the sum
c           continuously
```

```
      do 30000 index = 1, n
        do 3100 jj= 1, n
           temp(1,jj) =  r_rkgill(index)*pfirst(index,jj)
           temp(2,jj) =  r_rkgill(index)*psecond(index,jj)
           temp(3,jj) =  r_rkgill(index)*pthird(index,jj)
 3100     continue
c        **** Calculate the term of the Putzer sum depending on index.
c        ****  Then add to the first index-1 sums stored in expmatt.
        do 3200 ii = 1, 3
          do 3210 jj = 1, n
            expmatt(ii,jj) = expmatt(ii,jj) + temp(ii,jj)
 3210     continue
 3200   continue
30000  continue
c--> Initialize the u vector to zero.....
      do 50000 i = 1, n
        u(i) = dcmplx( 0.0D+00, 0.0D+00)
50000  continue
c-->  Form the coordinate vector for time t
      do 90100 i = 1, 3
        do 90110 j = 1, n
          u(i) = u(i) + expmatt(i,j)*u0(j)
90110   continue
90100  continue
      phi  = dreal(u(1)) + alpha*xlphi*t
      q    = dreal(u(2))
      p    = dreal(u(3))
      write(6,'(1x,D14.6,3x,D17.9,2x,D17.9,2x,D17.9)' ) t, phi, q, p
  40 continue
     write(*,'('' '',71(1H*))' )
     write(6,'('' '',71(1H*))' )
     close(6)
     go to 99999
 215 write(*,'('' EOF found in header of eigenvalue data file: '',A24)
    &') ev_file
     write(*,'('' '',71(1H*))' )
     write(6,'('' '',71(1H*))' )
     close(6)
     stop 00215
 225 write(*,'('' EOF found while reading eigenvalue data on file: '',
    &A24)') ev_file
     write(*,'('' '',71(1H*))' )
     write(6,'('' '',71(1H*))' )
     close(6)
     stop 00225
99999 continue
     end
c******************** END of PROGRAM PUTKEP6 ************************
```

```
$DEBUG
      subroutine crosspr(a,b,c)
c*************************************************************************
c******* This subroutine, provided with the vectors a and b, which ****
c******* have real components, performs double precision arithmetic ***
c******* to obtain the cross product of the vectors a and b.  The cross
c******* product operation is defined in the standard manner.  The ****
c******* result is stored in the vector c. All vectors have 3 component
c*************************************************************************
      implicit real*8 (a-h,o-z)
      dimension a(3), b(3), c(3)
      c(1)= a(2)*b(3) - b(2)*a(3)
      c(2)= a(3)*b(1) - b(3)*a(1)
      c(3)= a(1)*b(2) - b(1)*a(2)
      return
      end
c******************** END of SUBROUTINE CROSSPR ************************
$DEBUG
      double precision function dot(a,b)
c*************************************************************************
c******* This function, provided with the vectors a and b, which have**
c******* real components, performs double precision arithmetic to *****
c******* obtain the dot product of the vectors a and b.  The dot ******
c******* product operation in the standard manner. ********************
c******* All vectors are assumed to have 3 or less components. *******
c*************************************************************************
      implicit real*8 (a-h,o-z)
      dimension a(3), b(3)
      dot = a(1)*b(1) + a(2)*b(2) + a(3)*b(3)
      return
      end
c************* END of DOUBLE PRECISION FUNCTION DOT  ******************
$DEBUG
      subroutine buildp6(ceigval)
c*************************************************************************
c******* This subroutine builds the P matrix of the Putzer Algorithm. *
c******* Necessary eigenvalues are supplied via the calling line. The *
c******* range of complex matrix P is returned. The entire set of P ***
c******* matrices is calculated.  Real*8 arithmetic is used. **********
c*************************************************************************
      parameter(nmax=6)
      implicit real*8 (a-h,o-z)
      integer n
      complex*16 p(nmax,nmax), f(nmax,nmax), pold(nmax,nmax)
      complex*16 pfirst(nmax,nmax), psecond(nmax,nmax),pthird(nmax,nmax)
      complex*16 ca(nmax,nmax), cident(nmax,nmax)
      complex*16 ceigval(nmax)
      common /matrices/ ca, cident, n
```

```
      common /pmatrix/  pfirst, psecond, pthird
c-->  Initialize the storage area of the first three rows of the entire
c       range of P matrices.  REMEMBER, index is the subscript of the
c       P matrix.
      do 110 jj = 1, n
        do 120 index = 1, n
            pfirst(index,jj) = dcmplx(0.0D+00, 0.0D+00)
            psecond(index,jj) = dcmplx(0.0D+00, 0.0D+00)
            pthird(index,jj) = dcmplx(0.0D+00, 0.0D+00)
  120     continue
  110 continue
c-->    Define the first three rows of P sub 0 & store using index of 1
      pfirst(1,1) = dcmplx(1.0D+00, 0.0D+00)
      psecond(1,2) = dcmplx(1.0D+00, 0.0D+00)
      pthird(1,3) = dcmplx(1.0D+00, 0.0D+00)
c-->    Initialize the pold matrix to the complex identity matrix.
      do 400 jj = 1, n
        do 410 ii = 1, n
          pold(ii,jj) = cident(ii,jj)
  410   continue
  400 continue
c--->   Compute the remainder of the P matrices using recursion --------
      do 4000 index = 2, n
        do 500 jj = 1, n
          do 510 ii = 1, n
            if(ii.eq.jj) then
              f(ii,ii) = ca(ii,ii) - ceigval(index-1)
            else
              f(ii,jj) = ca(ii,jj)
            endif
  510     continue
  500   continue
c-->     Now compute P using the P matrix just calculated....
        do 600 ij = 1, n
          do 610 kj = 1, n
            do 700 kl = 1, n
              p(ij,kj) = p(ij,kj) + f(ij,kl)*pold(kl,kj)
  700       continue
  610     continue
  600   continue
c-->      Transfer P matrix to POLD matrix.....
        do 800 j = 1, n
          do 810 i = 1, n
            pold(i,j) = p(i,j)
  810     continue
  800   continue
c-->      Save first two rows of the P matrix sub INDEX for Putzer
c           calculations.......
```

```
      do 900 i = 1, n
        pfirst(index,i) = p(1,i)
        psecond(index,i) = p(2,i)
        pthird(index,i) = p(3,i)
  900   continue
c-->      Re-initialize P matrix for matrix multiplication loops.
      do 1200 k = 1, n
        do 1210 j = 1, n
          p(j,k) = dcmplx(0.0D+00, 0.0D+00)
 1210     continue
 1200   continue
 4000 continue
      return
      end
c********************** END OF SUBROUTINE BUILDP6 **********************
$DEBUG
      subroutine getrk6(h,n,cval,deltat,r_begin,r,begin_time,end_time)
c***************************************************************************
c******** This subroutine finds the numerical values of the polynomial *
c******** functions R sub K by performing Runge-Kutta integration upon *
c******** the entire set of N simultaneous first-order linear **********
c******** differential equations.  The integration is carried out from *
c******** BEGIN_TIME to the value contained in the variable END_TIME. **
c******** The variable R_BEGIN contains the values of R valid at the ***
c******** time BEGIN_TIME, and thus contain the initial conditions of **
c******** the integration.  ********************************************
c******** The array CVAL contains the set of complex eigenvalues for ***
c******** the truncation being solved.  The Runge-Kutta routine for N **
c******** simultaneous linear first-order equations is taken from ******
c******** Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c******** 1974 McGraw-Hill pp. 675-678 *********************************
c******** The system to be solved can be represented by the following **
c******** matrix equation. >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> ***
c********                                                               **
c********    -  -     -                                  -  -   -       **
c********    |     |   | cval(1) 0    0 ..............0 | | r(1) |      **
c********    |  _  |   |   1  cval(2) 0 ..............0 | | r(2) |      **
c********    | dr  | = |   0     1 cval(3) 0 ..........0 | | r(3) |      **
c********    | --  |   |   0     0    1 cval(4) 0 ......0 | |   .  |      **
c********    | dt  |   |   0     0    0   1 cval(5) 0...0 | |   .  |      **
c********    |     |   |   .     .    .   0  \   \  0 ..0 | |   .  |      **
c********    |     |   |   .     .    .   .    \   \ 0 .0 | |   .  |      **
c********    |     |   |   0     0    0   0  ..  1 cval(n)| | r(n) |      **
c********    -  -     -                                  -  -   -       **
c********                                                               **
c******** ------------------------------------------------------------ ***
c******** This subroutine builds only those polynomials for the  6x6  **
c******** truncation of the L matrix. ***********************************
```

```
c*************************************************************************
      parameter(nmax=6)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrol
      integer ul_steps, knt_steps
      complex*16  r(nmax), rdot(nmax), cval(nmax), r_begin(nmax)
      data t0/0.0D+00/
c-->  Initialize Runge routine variables, control index (m)
      m = 0
c-->  Determine the total number of steps (ul_steps) to make with Runge-
c       Kutta-Gill routine executing stepsize of H to advance solution
c         from begin_time to end_time.....
c       YY corresponds to hfactor...
      yy = deltat/h
      if(yy .lt. 1.0D+00) stop 00111
      dtt = deltat
      klim = idint(yy)
      do 4545 jh = 1, klim + 2
        dtt = dtt - h
        if((dtt - 0.0D+00) .le. 1.0D-08) go to 4550
 4545 continue
 4550 ul_steps = jh
c-->  Initialize step counter (knt_steps)......
      knt_steps = 0
c-->    Initialize the r polynomial vector to values at time
c                       t = begin_time
      do 2300 jj = 1, n
        r(jj) = r_begin(jj)
 2300 continue
      t = begin_time
c     ----------------------------------------------------------------
    8 if(knt_steps - ul_steps) 6,7,7
    6 continue
      call runge6(n,r,rdot,t,h,m,kontrol)
      go to (10,20) kontrol
c       Define first-order derivatives of the system....
   10 rdot(1) = cval(1)*r(1)
      do 3300 kj = 2, n
        rdot(kj) = r(kj-1) + cval(kj)*r(kj)
 3300 continue
      go to 6
   20 continue
      knt_steps = knt_steps + 1
      go to 8
    7 continue
      end_time = t
      return
      end
```

```
c******************** END OF GETRK6  ********************************
$DEBUG
      subroutine runge6(n,y,f,x,h,m,kontrl)
c*************************************************************************
c******** This subroutine performs Runge-Kutta integration by the Gills*
c******** method. The Runge-Kutta routine for N ************************
c******** simultaneous linear first-order equations is taken from ******
c******** Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c******** 1974 McGraw-Hill pp. 675-678  ********************************
c*************************************************************************
      parameter(nmax=6)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrl
      complex*16 y(nmax), f(nmax), q(nmax)
      m = m + 1
      go to (1,4,5,3,7) m
    1 do 2 i = 1, n
        q(i) = dcmplx(0.0D+00,0.0D+00)
    2 continue
      a = 0.5D+00
      go to 9
    3 a = 1.7071067811865475244D+00
    4 x = x + (.5D+00)*h
    5 do 6 i = 1, n
        y(i) = y(i) + a*(f(i)*h - q(i))
        q(i) = (2.0D+00)*a*h*f(i) + ((1.0D+00) - (3.0D+00)*a)*q(i)
    6 continue
      a = .29289321881345247560+00
      go to 9
    7 do 8 i = 1, n
        y(i) = y(i) + h*f(i)/(6.0D+00)  -  q(i)/(3.0D+00)
    8 continue
      m = 0
      kontrl = 2
      go to 10
    9 kontrl = 1
   10 continue
      return
      end
c****************** END OF SUBROUTINE RUNGE6  ************************
```

APPENDIX I

FORTRAN LISTING FOR RUNGE-KUTTA-GILL NUMERICAL INTEGRATION OF KEPLER PROBLEM

```
$DEBUG
      program solvekep
C
C*************************************************************************
C*************************************************************************
C********                                                          ***
C********   This program solves the Hamiltonian for the Kepler     ***
C********   problem in a plane  by integrating the resultant Hamilton's*
C********     equations using a generalized fourth order RUNGE-KUTTA ***
C********     technique employing GILL'S method. ********************
C********     The Runge-Kutta routine for N simultaneous linear      ***
C********     first-order equations is taken from Appendix C of the  ***
C********     book VISCOUS FLUID FLOW by FRANK M. WHITE 1974 McGraw- ***
C********     Hill pp. 675-678 which implements GILL'S method of     ***
C********     Runge-Kutta integration.  ********************************
C*************************************************************************
C********   The program prompts the user for a filename (DOS) to    ***
C********      which it will write the data.  As written, at most   ***
C********      500 data points will be written.                     ***
C********                                                          ***
C********   In MKS units...                                        ***
C********      H =  (Pr)*(Pr)/(2m*r0*r0) + ((lphi)**2)/(2*m*r0*r0*  ***
C********           (1 + Q)**2) - GMm/(r0*(1+Q))                   ***
C********                                                          ***
C********   In CANONICAL units....                                 ***
C********      H =  (Pr)*(Pr)/(2*r0*r0) + ((lphi)**2)/(2*r0*r0*    ***
C********           (1 + Q)**2) -  1/(r0*(1+Q))                    ***
C********                                                          ***
C********   The user is prompted for his choice of units to work in. ***
C********                                                          ***
C*************************************************************************
C*******  Units are in the MKS system. Kg, Km, sec OR a CANONICAL UNIT *
C*******  system Kg, Radius of earth in Km, CTU (Canonical Time Unit) **
C*******  = 806.5 sec = 13.44 min, at the choice of the user. **********
C*************************************************************************
C*******  Cylindrical coordinate system is assumed.  r, phi, z ********
C*************************************************************************
C*************************************************************************
C
      parameter(nmax=3)
      implicit real*8 (a-h,o-z)
```

```
      integer n, nsteps
      integer*2 ihr,iminute,isecond,i100th,iyr,imon,iday
      character*24 filename
      character*1  istepans
      character*1 i_pick_units
      character*8 GMunits
      character*2 Runits
      character*3 Tunits
      character*6 Vunits
      character*9 Punits
      character*10 AMunits
      logical canonical /.FALSE./
      logical ichgstep /.FALSE./
      logical initialze /.TRUE./
      dimension  y(3), y_last(3)
      dimension rinit(3), vinit(3), anominit(3), c(3)
c-->    Physical constant data....
      data convmkm/1.0D-03/
      data pi/3.1415927D+00/
      data G/6.67D-17/, earth_mas/5.98D+24/, r_earth/6.378165D+03/
      data ctu_per_sec/1.2398521D-03/
      data h/.10D+00/
c-->  Call GETDAT and GETTIM (obtain system date and time)
      call getdat (iyr,imon,iday)
      call gettim (ihr,iminute,isecond,i100th)
      n = nmax
      write(*,'('' Input the value of n in Kg:'',/,
     &''  Use the format:'',/,
     &'' D11.4'' ,/,
     &'' Sd.ddddD+XX'' )')
      read(*,1229) xm
 1229 format(1x,D11.4)
      write(*,'(''  Default coord system units is the MKS system.'')')
      write(*,'(''  Do you wish to use CANONICAL system of units? Enter
     &Y or N'')')
      read(*,'(A1)') i_pick_units
      if((i_pick_units .eq. 'Y') .or. (i_pick_units .eq. 'y')) canonical
     & =  .TRUE.
c-->    Input initial components for position vector....
      write(*,'('' Input initial values RINITIAL(radial) in Km: '',/,
     &''  Use the format:'',/,
     &'' D14.7'' ,/,
     &'' Sd.ddddddD+XX'' )')
      read(*,1231) rinit_radial
 1231 format(1x,D14.7)
 1232 format(1x,D14.7)
      write(*,1232) rinit_radial
      if( .not. (canonical) ) then
```

```
        GM = G*earth_mas*convmkm
        rinit(1) = rinit_radial
        rinit(2) = 0.0D+00
        rinit(3) = 0.0D+00
      else
        GM = 1.0D+00
        xm = 1.0D+00
        rinit(1) = rinit_radial/r_earth
        rinit(2) = 0.0D+00
        rinit(3) = 0.0D+00
      endif
c-->    Input initial components for momentum vector....
      write(*,'('' Input initial velocity values VINITIAL(radial), and V
     &INITIAL(transverse): in Km/sec '',/,
     &'' Use the format:'',/,
     &'' D14.7,1X,D14.7'' ,/,
     &'' Sd.ddddddd+XXbSd.ddddddd+YY'' )')
      read(*,1233) vinit_radial, vinit_trans
      write(*,1234) vinit_radial, vinit_trans
 1233 format(1x,D14.7,1x,D14.7)
 1234 format(1x,D14.7,1x,D14.7)
      if( .not. (canonical) ) then
        vinit(1) = vinit_radial
        vinit(2) = vinit_trans
        vinit(3) = 0.0D+00
      else
        vinit(1) = (vinit_radial/r_earth)/ctu_per_sec
        vinit(2) = (vinit_trans/r_earth)/ctu_per_sec
        vinit(3) = 0.0D+00
      endif
c-->    Input initial values for phi, q, and p......
      write(*,'('' Input initial values of PHI, radial deviation Q, and
     &P: in rad, unitless, & Kg-Km2/sec'',/,
     &'' Use the format:'',/,
     &'' D14.7,1X,D14.7,1X,D14.7'' ,/,
     &'' Sd.ddddddd+XXbSd.ddddddd+YYbSd.ddddddd+ZZ'' )')
      read(*,1235) phizero, qzero, pzero
      write(*,1236) phizero, qzero, pzero
 1235 format(1x,D14.7,1x,D14.7,1x,D14.7)
 1236 format(1x,D14.7,1x,D14.7,1x,D14.7)
      if(canonical) then
        pzero =  (pzero/(r_earth*r_earth))/ctu_per_sec
      endif
c-->     ...open file to save integrated values of y(1), y(2), & y(3)..
      write(*,'('' Please enter the filename to use in writing the outpu
     -t:'')' )
      read(*,160) filename
      open(6,file=filename,status='new')
```

```
c-->      Calculation of initial angular momentum....
      call crosspr(rinit,vinit,c)
      amominit(1) = xm*c(1)
      amominit(2) = xm*c(2)
      amominit(3) = xm*c(3)
c-->      Calculation of magnitude of initial angular momentum....
      xlphisq = dot(amominit,amominit)
      xlphi = dsqrt(xlphisq)
c-->      Calculation of circular orbit radius....
      if( .not. (canonical) ) then
        r0 = xlphisq/(GM*xm*xm)
        r0overre = r0/r_earth
      else
        r0 = xlphisq
        r0overre = r0
      endif
        a  = r0
c-->      Calculation of initial period using Kepler's Third Law
      if( .not. (canonical) ) then
        period = (2.0D+00*pi*a*dsqrt(a))/dsqrt(GM)
        period_min = period/60.0D+00
      else
        period = 2.0D+00*pi*a*dsqrt(a)
        period_min = period*(1.0D+00/ctu_per_sec)*(1.0D+00/60.0D+00)
      endif
c-->   Calculate the non zero eigenvalues of the 4x4 L matrix.....
      if( .not. (canonical) ) then
        alpha = 1.0D+00/(xm*r0*r0)
        alphasq = alpha*alpha
c       beta  = GM*xm/r0
        omega = (GM*GM)*xm*xm*xm/(xlphisq*xlphi)
        omegasq = omega*omega
      else
        alpha = 1.0D+00/(r0*r0)
        alphasq = alpha*alpha
c       beta = 1.0D+00/r0
        omega = 1.0D+00/(r0*dsqrt(r0))
        omegasq = 1.0D+00/(r0*r0*r0)
      endif
      write(*,'('' Input the init time, no of orbits, and no.of steps.''
     &,/,''  Use the format:'',/,
     &'' D10.4,1X,D10.4,1X,I5'' ,/,
     &'' Sd.dddD+XXbSd.dddD+YYb54321'' )')
      read(*,1245) tbegin, znorbits, nsteps
 1245 format(1x,D10.4,1x,D10.4,1x,I5)
c-->      Calculate the increment to advance the time with
      tend = znorbits*period
      deltat = (tend - tbegin)/dble(nsteps)
```

```
      t0 = tbegin
c         ....write data to output file for records ...
      if( .not. (canonical) ) then
        GMunits = 'Km3/sec2'
        Runits  = 'Km'
        Vunits  = 'Km/sec'
        Tunits  = 'sec'
        Punits  = 'Kg-Km/sec'
        AMunits = 'Kg-Km2/sec'
      else
        GMunits = 'Re3/CTU2'
        Runits  = 'Re'
        Vunits  = 'Re/CTU'
        Tunits  = 'CTU'
        Punits  = 'Kg-Re/CTU'
        AMunits = 'Kg-Re2/CTU'
      endif
      write(*,'(''  Size of DELTAT is:'',D12.6,1x,A3)') deltat, Tunits
      write(*,'(''  Want STEPSIZE .LE. DELTAT '')')
      write(*,'(''  Default value of integration STEPSIZE is:'',D10.4)')
     & h
      write(*,'(''  Do you wish to change it?  Enter (Y/N)'')')
      read(*,'(A1)') istepans
      if((istepans .eq. 'y') .or. (istepans .eq. 'y')) ichgstep = .TRUE.
      if(ichgstep) then
        write(*,'(1x,''Enter the value (>= 1) to divide DELTAT by to pro
     &duce the value for STEPSIZE, use format'')')
        write(*,'(1x,''Sd.ddddddd0+XX'' )')
        read(*,'(D14.7)') hfactor
        write(*,'(D14.7)') hfactor
        h = deltat/hfactor
      endif
c-->  Initialize Runge routine variables, control index (m)
      m = 0
      write(6,257)
      write(6,258)
      write(6,250)
      write(6,260)
      write(6,261) xm, GM, GMunits
      write(6,262) rinit(1), Runits
      write(6,263) vinit(1), Vunits, vinit(2), Vunits
      write(6,264) phizero, qzero, pzero
      write(6,250)
      write(6,265) (amominit(i), i = 1,3)
      write(6,266) xlphisq, AMunits, xlphi, AMunits
      write(6,267) r0, Runits, period, Tunits, period_min, znorbits
      write(6,268) r0overre
      write(6,269) omega
```

```
      write(6,250)
      write(6,270) nsteps
      write(6,271) tbegin, tend, Tunits, deltat, Tunits, h, Tunits
      write(6,272) filename
      write(6,273) imon, iday, iyr,  ihr, iminute, isecond
      write(6,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
C         ....write data to screen for verification ...
      write(*,257)
      write(*,258)
      write(*,250)
      write(*,260)
      write(*,261) xm, GM, GMunits
      write(*,262) rinit(1), Runits
      write(*,263) vinit(1), Vunits, vinit(2), Vunits
      write(*,264) phizero, qzero, pzero
      write(*,250)
      write(*,265) (anominit(i), i = 1,3)
      write(*,266) xlphisq, AMunits, xlphi, AMunits
      write(*,267) r0, Runits, period, Tunits, period_min, znorbits
      write(*,268) r0overre
      write(*,269) omega
      write(*,250)
      write(*,270) nsteps
      write(*,271) tbegin, tend, Tunits, deltat, Tunits, h, Tunits
      write(*,272) filename
      write(*,273) imon, iday, iyr,  ihr, iminute, isecond
      write(*,'('' +/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/+/'')')
  160 format(A24)
  250 format(1H ,'           ')
  257 format(1H ,' EXACT SOLUTION BY RUNGE-KUTTA GILL INTEGRATION.')
  258 format(1H ,' KEPLER PROBLEM in a PLANE, DEVIATIONS FROM CIRCULAR O
     &RBIT @.')
  259 format(1H ,'RUNNING 4TH ORDER RUNGE-KUTTA GILL INTEGRATION ON THE
     $DATA BELOW.')
  260 format(1H ,' SUMMARY OF DATA TO BE USED FOLLOWS:')
  261 format(1H ,' Sat XM is:',D11.4,' Kg    GM is:',D14.6,1x,A8)
  262 format(1H ,' Rinit_radial is:',D12.6,1x,A2
     &   ,/,10x,' (ABOVE USED IN CALC INITIAL ANGULAR MOMENTUM VECTOR)')
  263 format(1H ,' Vinit_radial is:',D12.6,1x,A6,1x,' Vinit_trans is:',D
     &12.6,1x,A6,/,10x,'(ABOVE USED IN CALC INITIAL ANGULAR MOMENTUM VEC
     &TOR)')
  264 format(1H ,' PHIZERO IS:',D12.6,' Rad  QZERO IS:',D12.6,' PZERO IS
     &:',D12.6,/,10x,' (ABOVE USED IN BUILDING INITIAL CONDITION VECTOR)
     &')
  265 format(1H ,' INITIAL ANGULAR MOMENTUM VECTOR IS:',3(D12.6,2x) )
  266 format(1H ,' MAGNITUDE SQ OF INITIAL ANG MOM VECTOR is:'
     &,D16.8,' (',A10,')**2',/,10x,' ANG MOM MAGNITUDE IS: ',D16.8,1x,
     & A10)
```

```
  267 format(1H ,' EQUILIBRIUM RADIUS RO IS:',D12.6,1x,A2,' PERIOD IS: '
     &,D12.6,1x,A3,/,10x,' PERIOD in MINUTES is: ',D12.6,3x,'No of ORBIT
     &S is: ',D12.6)
  268 format(1H ,' Equilbrium radius in EARTH RADII is: ',D12.6)
  269 format(1H ,' Eigenvalue omega (non-zero) is : ',D14.8)
  270 format(1H ,'INTEGRATION LIMITS FOLLOW;        NSTEPS IS ',I6)
  271 format(1H ,' TBEGIN = ',D12.6,' ; TEND = ',D12.6,1x,A3,' DELTAT =
     -',D14.7,1x,A3,/,5x,' and RKG STEPSIZE is: ',D14.7,1x,A3)
  272 format(1H ,' OUTPUT DATA IS CONTAINED ON THE FILE CALLED- ',A24)
  273 format(1H ,' THE DATA ON THE ABOVE FILENAME WAS PRODUCED ON ',I2,'
     --',I2,'-',I4,/,35x,' AT APPROXIMATELY ',I2,1H:,I2,1H:,I2)
c          .....initialize necessary variables
      n = nmax
      t0 = 0.0
      t = t0
c-----------------------------------------------------------------------
      do 44 k=1, nsteps + 1
        t_last = t
        if((t .eq. 0.0) .and. (initialze)) then
c         y(1) is PHI, y(2) is Q, y(3) is Pr
          y(1) = phizero
          y(2) = qzero
          y(3) = pzero
          initialze = .FALSE.
        else
c-->  Define last values of the y(n) vector for use as the
c         initial condition vector for the RKGill integration.
          y_last(1) = y(1)
          y_last(2) = y(2)
          y_last(3) = y(3)
c-->  For the time T, get the entire set of the Y(n)
c         functions, each evaluated at the time T, given that the set
c         has the initial conditions given by the
c         values contained within the vector Y_LAST(n).....
c
c     Pass DELTAT to determine the number of times to execute Runge-
c         Kutta-Gill routine
          call dorkg(n,xm,r0,GM,xlphisq,deltat,h,y_last,y,t_last,t)
c-----------------------------------------------------------------------
        endif
c-->  Write values to the output file......
c         y(1) is PHI, y(2) is Q, y(3) is Pr
        write(6,'(1x,D14.6,3x,D17.9,2x,D17.9,2x,D17.9)' ) t, y(1), y(2),
     &   y(3)
   44 continue
c       ....close output file.....
      write(*,'('' '',71(1h*))' )
      write(6,'('' '',71(1h*))' )
```

```
      endfile 6
      rewind 6
      close(6)
      end
c************************** END of SOLVEKEP ************************
$DEBUG
      subroutine dorkg(n,xm,r0,GM,xlphisq,dt,h,y_begin,y,t_begin,t_end)
c*******************************************************************
c*******                                                         ***
c******* This routine serves as the interface to the Runge-Kutta-Gill***
c*******  subroutine rkgill.  It controls the number of times that  ***
c*******  the runge steps of rkgill are executed. It is designed to  ***
c*******  advance the solution of y at time t_last, y_last, to the  ***
c*******  the solution valid at time t, given by y. ******************
c*******************************************************************
C*******************************************************************
      parameter(nmax=3)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrol
      integer deltat_steps, knt_steps
      dimension f(3), y(3), y_begin(3)
c-->  Determine the total number of steps (DELTAT_STEPS) to make with
c       Runge-Kutta-Gill routine executing stepsize of H to advance
c       solution by amount DELTAT, from T_BEGIN to T.....
      yy = dt/h
      if(yy .lt. 1.0) stop 00111
      dtt = dt
      klim = idint(yy)
c       Do loop used to get precise correct value for DELTAT_STEPS
      do 4545 jh = 1, klim + 2
        DTT = DTT - H
        if((dtt - 0.0D+00) .le. 1.0D-06) go to 4550
 4545 continue
 4550 deltat_steps = jh
c-->  Initialize Runge routine variables, control index (m)
      m = 0
c-->  Initialize step counter (knt_steps)......
      knt_steps = 0
      t = t_begin
c         y(1) is PHI, y(2) is Q, y(3) is Pr
      y(1) = y_begin(1)
      y(2) = y_begin(2)
      y(3) = y_begin(3)
c-------------------------------------------------------------------
c
c       ....Call on 4-th order Runge-Kutta-Gills function
    8 if(knt_steps - deltat_steps) 6,7,7
    6 continue
```

```
      call rkgill(n,y,f,t,h,m,kontrol)
      go to (10,20) kontrol
c       Define first-order derivatives of the system....
   10 f(1) = xlphisq/(xm*(r0*r0)*(1.0D+00 + y(2))*(1.0D+00 + y(2)) )
      f(2) = ( 1.0D+00/(xm*(r0*r0)) )*y(3)
      f(3) = ( xlphisq/(xm*(r0*r0)*(1.0D+00 + y(2))**3) ) - GM*xm/(r0*
     & (1.0D+00 + y(2))*(1.0D+00 + y(2)) )
      go to 6
   20 continue
      knt_steps = knt_steps + 1
      go to 8
    7 continue
      t_end = t
      return
      end
c********************  END of DORKG ***********************************
$DEBUG
      subroutine rkgill(n,y,f,x,h,m,kontrl)
c*************************************************************************
c*******  This subroutine performs Runge-Kutta integration by the Gills*
c*******  method.  The Runge-Kutta routine for N ************************
c*******  simultaneous linear first-order equations is taken from ******
c*******  Appendix C of the book VISCOUS FLUID FLOW by FRANK M. WHITE **
c*******  1974 McGraw-Hill pp. 675-678  *********************************
c*************************************************************************
      parameter(nmax=3)
      implicit real*8 (a-h,o-z)
      integer m, n, kontrl
      dimension  y(nmax), f(nmax), q(nmax)
      m = m + 1
      go to (1,4,5,3,7) m
    1 do 2 i = 1, n
        q(i) = 0.0D+00
    2 continue
      a = 0.5D+00
      go to 9
    3 a = 1.7071067811865475244D+00
    4 x = x + (.5D+00)*h
    5 do 6 i = 1, n
        y(i) = y(i) + a*(f(i)*h - q(i))
        q(i) = (2.0D+00)*a*h*f(i) + ((1.0D+00) - (3.0D+00)*a)*q(i)
    6 continue
      a = .29283218813452475 6D+00
      go to 9
    7 do 8 i = 1, n
        y(i) = y(i) + h*f(i)/(6.0D+00)  -  q(i)/(3.0D+00)
    8 continue
      m = 0
```

```
      kontrl = 2
      go to 10
    9 kontrl = 1
   10 continue
      return
      end
c****************  END OF SUBROUTINE RKGILL  ************************
$DEBUG
      subroutine crosspr(a,b,c)
c*************************************************************************
c*******  This subroutine, provided with the vectors a and b, which ****
c*******  have real components, performs double precision arithmetic ***
c*******  to obtain the cross product of the vectors a and b.  The cross
c*******  product operation is defined in the standard manner.  The ****
c*******  result is stored in the vector c. All vectors have 3 component
c*************************************************************************
      implicit real*8 (a-h,o-z)
      dimension a(3), b(3), c(3)
      c(1)= a(2)*b(3) - b(2)*a(3)
      c(2)= a(3)*b(1) - b(3)*a(1)
      c(3)= a(1)*b(2) - b(1)*a(2)
      return
      end
c*******************  END of SUBROUTINE CROSSPR ***********************
$DEBUG
      double precision function dot(a,b)
c*************************************************************************
c*******  This function, provided with the vectors a and b, which have**
c*******  real components, performs double precision arithmetic to *****
c*******  obtain the dot product of the vectors a and b.  The dot ******
c*******  product operation in the standard manner. ********************
c*******  All vectors are assumed to have 3 or less components. *******
c*************************************************************************
      implicit real*8 (a-h,o-z)
      dimension a(3), b(3)
      dot = a(1)*b(1) + a(2)*b(2) + a(3)*b(3)
      return
      end
c*************  END of DOUBLE PRECISION FUNCTION DOT  ******************
```

The two page vita has been removed from the scanned document. Page 1 of 2

The two page vita has been removed from the scanned document. Page 2 of 2