ELSEVIER

Computer Physics Communications 108 (1998) 240–258

Computer Physics Communications

POMULT: A program for computing periodic orbits in Hamiltonian systems based on multiple shooting algorithms

Stavros C. Farantos [1]

Institute of Electronic Structure and Laser, Foundation for Research and Technology, Hellas, Greece
Department of Chemistry, University of Crete, Iraklion, Crete 711 10, Greece

Received 20 November 1996; revised 30 October 1997

Abstract

POMULT is a FORTRAN code for locating Periodic Orbits and Equilibrium Points in Hamiltonian systems based on 2-point boundary value solvers which use multiple shooting algorithms. The code has mainly been developed for locating periodic orbits in molecular Hamiltonian systems with many degrees of freedom and it utilizes a damped Newton–Raphson method and a secant method. The Graphical User Interface has also been written in the tcl-tk script language for interactively manipulating the input and output data. POMULT provides routines for a general analysis of a dynamical system such as fast Fourier transform of the trajectories, Poincaré surfaces of sections, maximum Lyapunov exponents and evaluation of the classical autocorrelation functions and power spectra. © 1998 Elsevier Science B.V.

PACS: 95.10.Eg; 95.10.Fh; 33.15.Mt
Keywords: Molecular dynamics and spectra; Periodic orbits; Multiple shooting algorithm; Damped Newton–Raphson method

PROGRAM SUMMARY

Title of program: POMULT (Periodic Orbit MULTishooting)

Catalogue identifier: ADHG

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland

Licensing provisions: none

Computer for which the program is designed and others on which it is operable:
Computers: Tested on workstations HP-9000/735, IBM-7030/3CT, PC-Linux; *Installation:* IESL-FORTH, Iraklion, Crete, Greece

Operating systems under which the program has been tested: UNIX

Programming language used: FORTRAN 77 with extensions, lower case, implicit, include, tcl-tk

Memory required to execute with typical data: 5 Mbytes

No. of bits in a word: 32

No. of bytes in distributed program, included test data, etc.: 1279057

Distribution format: uuencoded compressed tar file

[1] E-mail: farandos@iesl.forth.gr

PII S0010-4655(97)00131-8

Keywords: molecular dynamics and spectra, periodic orbits, multiple shooting algorithm, damped Newton–Raphson method

Nature of the physical problem
Given a multidimensional highly coupled molecular potential energy surface, we want to compute families of periodic solutions of Hamilton equations. These families of periodic orbits reveal the structure of the classical phase space by detecting the regions of phase space with regular and chaotic motions. Furthermore, periodic orbits point out possible localization of the quantum wavefunctions, and explain/predict spectroscopic features.

Method of solution
The location of periodic orbits is based on damped Newton–Raphson methods or secant-Quasi Newton methods. Simple or Multiple shooting algorithms are employed which are robust in cases of long period or highly unstable periodic orbits.

Restrictions on the complexity of the problem
The program has been tested with 2-, 3-, 5-, and 6-dimensional molecular potential functions. Limitations are observed in cases of high instability or in regions of phase space densely occupied by periodic orbits. The above difficulties cause also limitations in the continuation of a family of periodic orbits with a parameter.

Typical running time
This depends on the complexity of the potential function, the period and the number of periodic orbits which are computed, and whether the equations of motion are stiff or not.

Remarks
Standard numerical actions like integration of ordinary differential equations and solution of linear algebraic equations are carried out with routines from the package "Numerical Recipes". The program can be interfaced with ODESSA or other available programs which carry out sensitivity analysis of differential or algebraic equations. Generally, the program has been written in such a way that the user can incorporate his/her own favorable subroutines. A Makefile, a README file as well as a help file are provided for the installation of the program and the explanation of the input data. The Graphical User Interface for the input data has been written in the tcl-tk script language. The user should ensure that the library versions tcl7.0 and tk4.0 or higher are installed on her/his system.

LONG WRITE-UP

1. Introduction

The application of nonlinear mechanics to molecular systems and particularly the study of vibrational spectroscopic resonances with periodic orbits has increased the interest of having robust algorithms to locate periodic orbits (POs) in Hamiltonian systems [1–4]. The location of POs sometimes may be a difficult task, since the molecular potential energy surfaces are multiple dimensional functions and the strong coupling among the degrees of freedom introduces large instabilities [5].

The problem of finding periodic orbits may be seen as a *2-point boundary value problem*. The boundary conditions are the relations of closing the trajectory in phase space after the period of time T,

$$\boldsymbol{q}(T) = \boldsymbol{q}(0)\,, \qquad \boldsymbol{p}(T) = \boldsymbol{p}(0)\,. \tag{1}$$

$\boldsymbol{q} = (q_1, q_2, \ldots, q_N)^+$ are the generalized coordinates of a dynamical system of N degrees of freedom, and $\boldsymbol{p} = (p_1, p_2, \ldots, p_N)^+$ their conjugate momenta. $^+$ denotes a column vector.

Several numerical methods have been proposed for solving in general 2-point boundary value problems [6]. The *shooting methods* convert the 2-point boundary value problem to an *initial value* one. Choosing an initial value for the trajectory, we integrate the equations of motion for the period of time T and check the discrepancy in the boundary conditions. By varying the initial conditions or some free parameters we successively approach the trajectory which satisfies the boundary conditions.

Another class of 2-point boundary value solvers includes the *relaxation methods*. In these methods the differential equations are replaced with difference equations by choosing an appropriate mesh of points for the variables. Then, starting with an approximate solution we try to bring it into successively closer agreement with the finite difference equations, and with the boundary conditions.

Collocation methods with cubic spline interpolation of the periodic solutions of autonomous ordinary differential equations have also been proposed [7,8].

Shooting and relaxation techniques have been applied to locate periodic orbits in molecular systems or model potentials. The shooting methods are the most popular [9]. The *Monodromy Method* of Baranger and coworkers [10–12] is a technique which is classified in the relaxation methods. Programs based on the collocation method are available such as the package of programs AUTO [13,14].

An extension of the shooting techniques which tries to incorporate the benefits of the relaxation algorithms is the *multiple shooting method* [15–20]. In this case, the 1-point initial value problem is converted to $(m-1)$ initial value problems by choosing m nodes along the period of time. We do not take a finite difference representation of the equations of motion, but instead we integrate $(m-1)$ trajectories and by varying their $(m-1)$ initial conditions we approach to a smooth trajectory which satisfies the closure conditions (Eqs. (1)).

The program presented here, POMULT, is an implementation of the multiple shooting algorithm which uses global convergence techniques in the Newton–Raphson iterations. The package is also enriched with tools for calculating Power Spectra with Fast Fourier Transform (FFT), Maximum Lyapunov Exponents [21] (MLE), the Classical Correlation Function [22] (CCF), and the construction of Poincaré Surfaces of Section [21] (PSS) for the exploration of the geometry of phase space of a dynamical system. A Graphical User Interface is provided for an easy and fast way to alter the input data.

2. Computational methods

For the simplification of the mathematical equations, we define the column vector

$$\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{p})^{+} . \tag{2}$$

Using $\boldsymbol{x}$, we can write Hamilton equations in the form

$$\frac{d\boldsymbol{x}(t)}{dt} = \dot{\boldsymbol{x}}(t) = J\nabla H[\boldsymbol{x}(t)] \quad (0 \le t \le T) , \tag{3}$$

where H is the Hamiltonian function, and J is the symplectic matrix

$$J = \begin{pmatrix} 0_N & I_N \\ -I_N & 0_N \end{pmatrix} . \tag{4}$$

0_N and I_N are the zero and unit $N \times N$ matrices, respectively. $J\nabla H(\boldsymbol{x})$ is a vector field, and J satisfies the relations

$$J^{-1} = -J \quad \text{and} \quad J^2 = -I_{2N} . \tag{5}$$

To find periodic solutions, it is necessary to solve Eqs. (3) subject to the 2-point boundary conditions,

$$\boldsymbol{x}(T) - \boldsymbol{x}(0) = 0 . \tag{6}$$

The above boundary value problem is converted to an *initial value problem* by considering the initial values of the coordinates and momenta, $\boldsymbol{s}$,

$$\boldsymbol{x}(0) = \boldsymbol{s} , \tag{7}$$

as independent variables in the nonlinear functions

$$\boldsymbol{B}(\boldsymbol{s}; T) = \boldsymbol{x}(T; \boldsymbol{s}) - \boldsymbol{s} . \tag{8}$$

$\boldsymbol{B}$ parametrically depends on the period T. We denote the roots of Eqs. (8) as $\boldsymbol{s}_*$, i.e.,

$$\boldsymbol{B}(\boldsymbol{s}_*; T) = 0 . \tag{9}$$

Hence, if s is a nearby value to the solution s_*, we can compute the functions $\boldsymbol{B}(s)$ by integrating Hamilton equations for the period T. By appropriately modifying the initial values s, we hope to converge to the solution, that is, $s \to s_*$ and $B \to 0$.

A common procedure to find the roots of Eq. (9) is the Newton–Raphson method. This is an iterative scheme and at each iteration, k, we update the initial conditions of the orbit,

$$s_{k+1} = s_k + \Delta s_k \,. \tag{10}$$

The corrections Δs_k are obtained by expanding Eqs. (8) in a Taylor series up to the first order

$$\begin{aligned} &\boldsymbol{B}(s_{k+1};T) \approx \boldsymbol{B}(s_k;T) + \frac{\partial \boldsymbol{B}}{\partial s_k}\Delta s_k = 0\,, \\ &\boldsymbol{B}(s_k;T) + \left[\frac{\partial \boldsymbol{x}_k(T;s_k)}{\partial s_k} - I_{2N}\right]\Delta s_k = 0\,, \end{aligned} \tag{11}$$

where at the kth iteration

$$\boldsymbol{B}(s_k;T) = \boldsymbol{x}_k(T;s_k) - s_k \,. \tag{12}$$

The matrix (Jacobian)

$$Z_k(T) = \frac{\partial \boldsymbol{x}_k(T;s_k)}{\partial s_k} \tag{13}$$

is the *Fundamental Matrix*, which is evaluated by integrating the *variational equations*

$$\dot{\boldsymbol{\zeta}}(t) = A(t)\boldsymbol{\zeta}(t) \quad (0 \le t \le T)\,, \tag{14}$$

where the second derivatives of the Hamiltonian with respect to coordinates and momenta are needed,

$$A(t) = J\partial^2 H[\boldsymbol{x}(t)]\,. \tag{15}$$

These equations calculate the linearized part of the difference of two initially neighboring trajectories in time, $\boldsymbol{\zeta} = \boldsymbol{x}' - \boldsymbol{x}$. The Fundamental Matrix is also a solution of the variational equations

$$\dot{Z}(t) = A(t)Z(t)\,. \tag{16}$$

Thus, to perform the kth iteration in the Newton–Raphson method, we first integrate for time T the differential equations

$$\begin{aligned} &\dot{\boldsymbol{x}}_k(t) = J\nabla H[\boldsymbol{x}_k(t)]\,, \\ &\dot{Z}_k(t) = A_k(t)Z_k(t)\,, \end{aligned} \tag{17}$$

with initial conditions

$$\begin{aligned} &\boldsymbol{x}_k(0) = s_k\,, \\ &Z_k(0) = I_{2N}\,. \end{aligned} \tag{18}$$

Then, we solve the linear algebraic equations

$$[Z_k(T) - I_{2N}]\Delta s_k = -\boldsymbol{B}(s_k;T)\,, \tag{19}$$

in order to find the initial conditions for the $(k+1)$th iteration (Eqs. (10)).

For a periodic orbit, the Fundamental Matrix at the time $t = T$ is called *Monodromy Matrix*, $M = Z(T)$. The eigenvalues of the Monodromy Matrix are used for the stability analysis of the periodic orbit. Details can be found in Ref. [19].

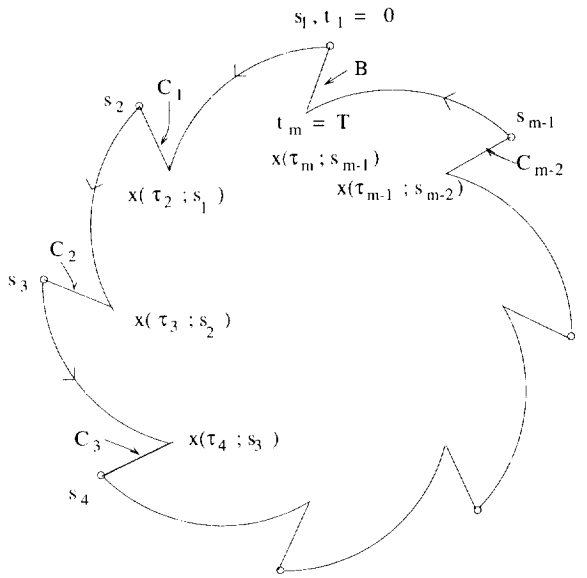

Fig. 1. A schematical representation of the multiple shooting procedure.

2.1. The multiple shooting method

Let us assume that we divide the period T in $(m-1)$ time intervals by choosing m nodes. The idea of the multiple shooting method is to integrate $m-1$ trajectories – one for each time interval – and to check if the final values of each trajectory coincide with the initial conditions of the next. To formulate this idea, we introduce the scaled time, $\tau = t/T$, $(0 \le \tau \le 1)$,

$$0 = \tau_1 < \tau_2 < \cdots < \tau_{m-1} < \tau_m = 1\,. \tag{20}$$

Thus, for the simple shooting method $m = 2$.

In this section we drop the index for the iterations k, and we use the index j to denote the nodes in the periodic orbit. If the initial conditions of the trajectory at each node j is $\boldsymbol{s}_j$ at time τ_j, and the final value of the trajectory at time τ_{j+1} is denoted by $\boldsymbol{x}(\tau_{j+1}; \boldsymbol{s}_j)$, then $(m-2)$ *continuity conditions* should be satisfied (for a graphical representation, see Fig. 1),

$$\boldsymbol{C}_j(\boldsymbol{s}_j, \boldsymbol{s}_{j+1}; T) = \boldsymbol{x}(\tau_{j+1}; \boldsymbol{s}_j) - \boldsymbol{s}_{j+1} = 0\,, \quad j = 1, 2, \ldots, m-2\,, \tag{21}$$

together with the boundary conditions

$$\boldsymbol{B}(\boldsymbol{s}_{m-1}, \boldsymbol{s}_1; T) = \boldsymbol{x}(\tau_m; \boldsymbol{s}_{m-1}) - \boldsymbol{s}_1 = 0\,. \tag{22}$$

Now, we have to solve $(m-1)$ initial value problems, and for that we adopt the Newton–Raphson method

$$\boldsymbol{C}_j(\boldsymbol{s}_j, \boldsymbol{s}_{j+1}; T) + \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{s}_j}\Delta \boldsymbol{s}_j + \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{s}_{j+1}}\Delta \boldsymbol{s}_{j+1} = 0\,. \tag{23}$$

These equations become

$$\boldsymbol{C}_j(\boldsymbol{s}_j, \boldsymbol{s}_{j+1}; T) + Z_j(\tau_{j+1})\Delta \boldsymbol{s}_j - \Delta \boldsymbol{s}_{j+1} = 0\,, \quad 1 \le j \le m-2\,. \tag{24}$$

Using the boundary conditions (Eqs. (22)), we get

$$\boldsymbol{B}(\boldsymbol{s}_{m-1}, \boldsymbol{s}_1; T) + Z_{m-1}(\tau_m)\Delta \boldsymbol{s}_{m-1} - \Delta \boldsymbol{s}_1 = 0\,, \tag{25}$$

where

$$Z_j(\tau_{j+1}) = \frac{\partial \boldsymbol{x}(\tau_{j+1}; \boldsymbol{s}_j)}{\partial \boldsymbol{s}_j} . \tag{26}$$

Eqs. (24), (25) are written in a matrix form of dimension $2N(m-1) \times 2N(m-1)$,

$$\begin{bmatrix} Z_1 & -I_{2N} & 0 & \cdots & 0 & 0 \\ 0 & Z_2 & -I_{2N} & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & Z_{m-2} & -I_{2N} \\ -I_{2N} & 0 & 0 & \cdots & 0 & Z_{m-1} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{s}_1 \\ \Delta \boldsymbol{s}_2 \\ \cdots \\ \Delta \boldsymbol{s}_{m-2} \\ \Delta \boldsymbol{s}_{m-1} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \\ \cdots \\ \boldsymbol{C}_{m-2} \\ \boldsymbol{B} \end{bmatrix} . \tag{27}$$

The above system of linear equations is solved by invoking the so-called *condensing algorithm* [17],

$$\Delta \boldsymbol{s}_1 = -E^{-1}\boldsymbol{u} , \tag{28}$$

$$\Delta \boldsymbol{s}_{j+1} = Z_j \Delta \boldsymbol{s}_j + \boldsymbol{C}_j , \quad j = 1, 2, \ldots, m-2 , \tag{29}$$

where

$$E = Z_{m-1} Z_{m-2} \cdots Z_2 Z_1 - I_{2N} ,$$

$$\boldsymbol{u} = \boldsymbol{B} + Z_{m-1}(\boldsymbol{C}_{m-2} + Z_{m-2}(\boldsymbol{C}_{m-3} + Z_{m-3}(\boldsymbol{C}_{m-4} + \ldots (\ldots + Z_2 \boldsymbol{C}_1) \ldots))) . \tag{30}$$

2.2. The damped Newton–Raphson method

Quite often the Newton–Raphson method diverges, although when it converges it does that quadratically. Problems of divergences are cured by scaling the Newton step with a parameter λ_k,

$$\boldsymbol{s}_{k+1} = \boldsymbol{s}_k + \lambda_k \Delta \boldsymbol{s}_k , \tag{31}$$

where $0 \leq \lambda_k \leq 1$, and $\lambda_k \to 1$ as $\boldsymbol{s}_k \to \boldsymbol{s}_*$. Newton methods of this type are called *damped or underrelaxed.*

Several schemes for choosing λ_k have been proposed [6,16]. One is to require that the selected step decreases the function

$$f = \tfrac{1}{2}\boldsymbol{B}(\boldsymbol{s}; T) \cdot \boldsymbol{B}(\boldsymbol{s}; T) . \tag{32}$$

It can be shown that the Newton step, $\Delta \boldsymbol{s}_k$, is a descent direction for f. Therefore, the strategy is first to test if the Newton step decreases sufficiently the function f. If not, we reduce the Newton step by a factor λ_k. A method to choose the appropriate λ_k is by backtracking along the Newton direction [6]. The method consists of modeling the function

$$g(\lambda_k) = f(\boldsymbol{s}_k + \lambda_k \Delta \boldsymbol{s}_k) \tag{33}$$

by a quadratic or cubic polynomial of λ_k, and then to find the value of λ_k that minimizes g. Details can be found in Numerical Recipes [6].

The linear system of Eqs. (19), $\Delta \boldsymbol{s}_k$, is solved by LU-Decomposition or Singular Value Decomposition methods. The convergence criteria that we use in the Newton–Raphson iteration scheme are

$$||\boldsymbol{B}(\boldsymbol{s}_k)|| = \max\left(|B_i(\boldsymbol{s}_k)|\right) < d_1 , \quad \text{Level 1} , \tag{34}$$

and

$$||\Delta \boldsymbol{s}_k|| = \max\left(|\Delta s_{i(k)}|\right) < d_2 , \quad \text{Level 2} . \tag{35}$$

d_1 and d_2 are preselected accuracy criteria.

2.3. The secant method

Another method that we have applied for solving Eqs. (9) is a quasi Newton–Raphson type procedure for the approximate evaluation of the Fundamental Matrix. Instead of an exact calculation of Z at each iteration k, we can use Broyden updating formula [6]

$$P_{k+1} = P_k + \frac{(\Delta \boldsymbol{B}_k - P_k \cdot \Delta \boldsymbol{s}_k) \otimes \Delta \boldsymbol{s}_k}{\Delta \boldsymbol{s}_k^+ \cdot \Delta \boldsymbol{s}_k}, \tag{36}$$

where $P_k = Z_k - I_{2N}$, $\Delta \boldsymbol{B}_k = \boldsymbol{B}_{k+1} - \boldsymbol{B}_k$, and $\Delta \boldsymbol{s}_k = \boldsymbol{s}_{k+1} - \boldsymbol{s}_k$. $\otimes$ denotes a direct product.

For the first value of P_0 it is advisable to start with an accurate evaluation of the Fundamental Matrix. In the algorithm described in Ref. [6], the *secant Broyden method* is combined with the backtracking scheme for global convergence. Then, a QR decomposition method is used to solve the linear system of equations at each iteration. Details can be found in Numerical Recipes [6].

2.4. Implementation

POMULT requires as input data an initial guess for the coordinates, momenta and the period of periodic orbit. In a multiple shooting calculation with m nodes, we produce $m-2$ more sets of initial data by integrating the initial trajectory and storing the coordinates and momenta at the times

$$t_i = \frac{i-1}{m-1} T, \quad i = 2, \ldots, m-1. \tag{37}$$

The simple and multiple shooting methods described up to now assume that the period of periodic orbit T is kept fixed through the iteration process and it is used as a parameter during the continuation of the particular family of periodic orbits which is studied. In this way of seeking a periodic orbit there is an arbitrariness as far as the initial conditions of the periodic orbit are concerned, since every point along the periodic orbit is a solution of Eq. (19). A method to eliminate such an arbitrariness is to keep fixed one of the coordinates or momenta. A practical way to implement this is to consider the period as a variable and to increase the boundary conditions by the equation

$$B^{2N+1} = x_l(0) - \xi = 0, \quad 1 \le l \le N, \tag{38}$$

or

$$B^{2N+1} = \dot{x}_l(0) = 0, \quad 1 \le l \le N. \tag{39}$$

l is the particular coordinate which is kept fixed or it takes an extremum value at $t = 0$. The equations of motion (Eq. (3)) are also increased by the trivial differential equation [19]

$$\dot{T} = 0. \tag{40}$$

Hence, a $(2N+1)$-dimensional boundary value problem must be solved and the corresponding equations are

$$\begin{bmatrix} Z_k - I_{2N} & \dot{\boldsymbol{x}}(0) \\ 1_l & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{s}_k \\ \Delta T_k \end{bmatrix} = - \begin{bmatrix} \boldsymbol{B}_k \\ B_k^{2N+1} \end{bmatrix}. \tag{41}$$

1_l denotes the result of the differentiation of Eq. (38) with respect to the lth initial condition.

The above method brings all periodic orbits on a common Poincaré plane of section. In the POMULT program we have also implemented the Henon method [23] to compute Poincaré surfaces of sections for arbitrary trajectories.

It is also possible to locate POs at specific total energy E by varying the period and using as an extra boundary condition the conservation of the total energy,

$$B^{2N+1} = H[\boldsymbol{x}(0)] - E = 0 . \tag{42}$$

In this case the $(2N+1)$th row in Eqs. (41) is replaced by the row $(-\dot{p}_1 \; -\dot{p}_2 \; \cdots \; -\dot{p}_N \; \dot{q}_1 \; \dot{q}_2 \; \cdots \; \dot{q}_N \; 0)$.

The $(m-1)$ Fundamental Matrices required in the multiple shooting method may be evaluated either from numerically obtained derivatives or analytically. The first requires the integration of $2N(m-1)$ neighboring trajectories, and the derivatives are then computed by finite differences. In the case that the analytic second derivatives of the Hamiltonian are available, we integrate Hamilton and the variational equations, Eqs. (17), (18), simultaneously.

For the stability analysis of a periodic orbit, we have to compute the Monodromy Matrix [24]. This means that we must integrate the variational equations of the periodic orbit for a full period T. However, an estimate of the Monodromy Matrix can be obtained from the product of Fundamental Matrices as are evaluated in the multiple shooting procedure

$$M = Z_{m-1} Z_{m-2} \cdots Z_2 Z_1 . \tag{43}$$

This method does not give accurate eigenvalues and generally is not recommended.

To obtain the whole family of periodic orbits at several energies (*continuation* of the family), we found it useful to use the period as the varying parameter. Continuation is carried out by predictor–corrector techniques with a trivial

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_0 , \tag{44}$$

or a secant predictor [25]

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_1 + \mu(\boldsymbol{x}_1 - \boldsymbol{x}_0) . \tag{45}$$

μ is a scaling factor, and $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are the initial conditions of previous converged POs.

3. Program structure

The subroutines are organized in six directories: *main, root, anals, integr, lineq*, and *dat_tcl*. Fig. 2 presents the structure of the program POMULT.

3.1. Directory 'main'

The directory *main* contains the MAIN program that reads the input data, calls the root finding routines, calls for the analysis of the PO, and decides for the continuation scheme. In the same directory the subroutines (DEEVAL, VARSEC, JAC, JACOBN) that interface the integrators of the differential equations with the *user supplied routines* (POTEN, SECDER) are also stored. DEEVAL calls the routine with Hamilton equations (POTEN), and VARSEC, JAC, and JACOBN call the routine with the derivatives of Hamilton equations (SECDER), if the second derivatives of the potential are available. Otherwise, SECDER should be a dummy routine.

The subroutines which calculate the Fundamental (or Monodromy) Matrix (MONOD, MONODB), the boundary conditions (BRC), and the continuation scheme (CONTIN) are also stored here. MONODB gives an approximate Monodromy Matrix according to Eq. (43).

The routine START computes the initial conditions for the $m-2$ nodes given a starting point in phase space. The routine CH2D checks analytic with numeric second derivatives.

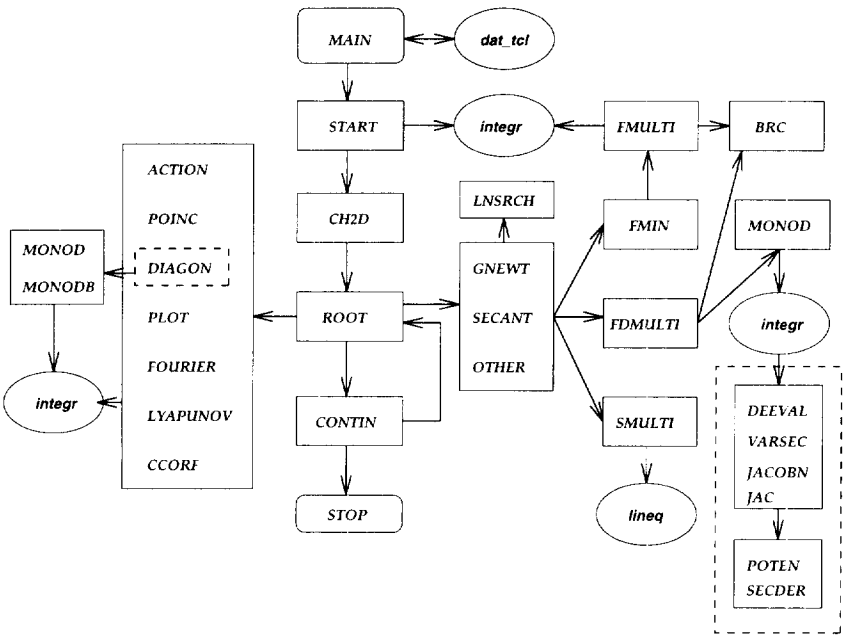

Fig. 2. A diagram which shows the subroutines of the program POMULT and their interconnectivity. The oval type graphs describe directories that contain routines for integrating Hamilton and variational equations (*integr*), for solving linear algebraic equations (*lineq*), and tcl-tk scripts for GUI (*dat_tcl*).

3.2. Directory 'root'

ROOT is the routine which decides about the type of solver in the root finding process. GNEWT is the dumped Newton–Raphson solver, and SECANT is the secant-Broyden one. LNSRCH is the routine used for backtracking along the Newton direction.

FMIN calculates the function f (Eq. (32)), FMULTI computes the functions $\boldsymbol{C}$ and $\boldsymbol{B}$ by using the continuity and boundary conditions, Eqs. (24), (25). FDMULTI computes the Fundamental Matrices. Finally, SMULTI computes the right-hand side of Eq. (27) and solves the linear system of equations, Eqs. (28), (29). OTHER is a dummy routine which can be used to interface this program with user-supplied root finders.

3.3. Directory 'anals'

In this directory we store all the subroutines needed for the analysis of PO. DIAGON diagonalizes the Monodromy Matrix for the stability analysis. ACTION computes the action integral along the PO. PLOT stores points for plotting the PO. POINC computes Poincaré Surfaces of section with the Henon method [23].

Finally, FOURIER carries out a FFT of a stored trajectory, and LYAPUNOV estimates the Maximum Lyapunov Exponent. CCORF is a set of routines which calculate the classical autocorrelation function starting with Gaussian distributions for the initial conditions of the trajectories.

3.4. Directory 'integr'

In this directory we store a number of integration routines for the first order differential equation, ranged from Runge–Kutta to Adams predictor–corrector algorithms. Routines for stiff equations are also included.

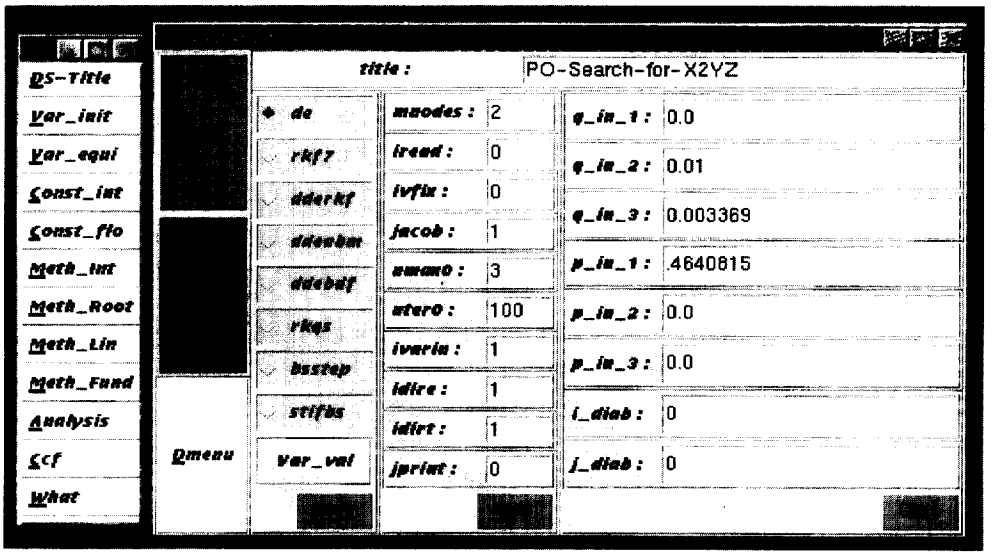

Fig. 3. The interface window for the definition of input data in the program POMULT.

3.5. Directory 'lineq'

In this directory we store routines for solving linear systems of equations based on LU, QR, and SVD (Singular Value Decomposition) methods. Routines for diagonalizing a matrix are found here.

3.6. Directory 'dat_tcl'

The directory *dat_tcl* keeps the tcl-tk scripts [26,27] for constructing windows to interactively manipulate the input/output data of the POMULT program (PO_T.DATTCL), see Fig. 3. This program should be edited by the user in order to define the location of the input file (po_t.dat) for POMULT.

3.7. Include files

There are two include files: the PARAMETER.INC and CONSTANTS.INC. To install a new application, first edit the PARAMETER.INC file to define the dimension of phase space *ndiml*, the maximum number of nodes (*mdiml*) along a periodic orbit as well as the maximum number of iterations in the computation of the Maximum Lyapunov Exponent (*mlen*), the size of the Fourier series (*mitt*) in the FFT analysis of the trajectories and the maximum integration time of the trajectories in the Classical Autocorrelation Functions (*mxtim*). Then, edit the file MAKEFILE_MAIN to define the directory where the subroutines with the potential function and its derivatives (POTEN) and possibly the second derivatives (SECDER) are stored. The executable program will be put in the same directory.

In CONSTANTS.INC file the values of some constants, units and conversion factors are given.

4. Input–output description

4.1. Open files

The following files are opened in the program.

Subroutine MAIN

Channel	Filename	Description
1	po_t.dat	Input data.
2	po_t.out	Output data.
3	po_t.int	Initial values for the nodes of the trajectory (to be read if iread – see below – not equal to 0).
9	po_t.nodes	Store initial values for all nodes of the periodic orbit for continuation; if iread not equal to zero then copy this file to po_t.int; if jprint – see below – greater than 0, store the initial values for all nodes of all computed POs.
10	po_t.res	Store several output data to be used for plotting the continuation/bifurcation diagrams.
11	po_t.wro	Store the Monodromy matrix of the final PO.

Subroutine POINC

Channel	Filename	Description
4	po_t.pss	Poincare Surface of section data.

Subroutine PLOT

Channel	Filename	Description
8	po_t.plo	Store time, total energy, potential energy, coordinates and momenta of the PO or of an arbitrary trajectory for plotting.

Subroutine FOURIER

Channel	Filename	Description
20	po_t.fourie-out	Store frequency and intensity only if the intensity is greater than a tolerance value -toler-.
30	po_t.fourie	Store all computed frequencies and intensities for plotting the spectrum.

Subroutine LYAPUNOV

Channel	Filename	Description
40	po_t.lyapunov	Store the rate of exponential divergence as a function of time.

Subroutine CCORF

Channel	Filename	Description
80	po_t.ccf	Output from classical correlation analysis.
81	po_t.cor	Store the autocorrelation functions of the selected variables.
83	po_t.spc	Store the FFT results of the autocorrelation functions.

4.2. Input data

All input data are given in free format. They are grouped in categories and the group name is that which appears in the menu widget (Fig. 3). Here is their definition.

DS_TITLE

title:	Title for the studied dynamical system.

VAR_INIT

xin(1:ndiml):	Starting point in phase space; give coordinates first and then conjugate momenta.
idiab:	Locate periodic orbits in a reduced dimension space.
jdiab(1:idiab):	The coordinates that will be excluded.

VAR_EQUI

xequ(1:ndiml): Equilibrium values of the coordinates and conjugate momenta, usually used for the definition of the variable that will be kept fixed or to define the Poincare plane of section.

iang: The number of angle coordinates.

jang(1:iang): Put the position numbers of the angle coordinates in the coordinate list; i.e. 3, if the third coordinate is the angle.

CONST_INT

mnodes: The number of nodes along the periodic orbit;
for simple shooting, mnodes = 2;
for multiple shooting, mnodes = mm.

iread: Decide to read or not the initial point and the nodes of the trajectory from the file "po_t.int";
0 no read;
1 yes read.

ivfix: Decide to keep or not one variable (ivar) fixed at the value "xequ(ivar)" or to keep the total energy constant;
0 no fixed value, the period does not change;
$+n$ fixed value "xc(ivar) = xequ(ivar)" ($1 \leq n \leq$ ndiml/2);
$-n$ the total energy is conserved;
when ivfix.ne.0, then the period T is also a variable in the Newton–Raphson iteration scheme.

jacob: Decide to keep the Jacobian constant or not in the Newton–Raphson iterations;
0 keep the Jacobian constant after the first iteration;
n vary the Jacobian at each iteration step ($n > 0$).

nmax: The total number of periodic orbits to be computed;
0 no location of POs, but only analysis of the current trajectory.

nter: The maximum number of iterations in the Newton–Raphson procedure.

ivarin: ivar = ivarin. The variable which defines the plane of section or the variable which will remain fixed at xequ(ivar) when ivfix $\neq$ 0.

idire: Decide for forward or backward continuation;
$+n$ forward;
$-n$ backward.

idirt: Decide for forward or backward integration in time;
$+n$ forward;
$-n$ backward.

jprint: Decide about the amount of output to be directed in the "@stdout";
-1 no printing at all;
0 initial data – iterative values of level functions – solution data (or final data, respectively);
$+1$ more output.

CONST_FLO

deltat: Parameter increment for the continuation of the family of POs;
if period is used for the continuation, then deltat is the step to increase (idire.gt.0) or to decrease the period;
if energy is used as a continuation parameter, then deltat denotes the energy step.

tstep: Integration time step for collecting points in phase space. Usually used in the analysis of the trajectories, FOURIER, LYAPUNOV and CCF.

period: The period of PO.
tolf: Convergence criterion for minimizing the functions (Level1).
tolx: Convergence criterion for the solution of the linear systems in Newton–Raphson method (Level2).
tol: Criterion used in the gnewt and secant subroutines to determine if convergence to a stationary point of the potential has occurred. It is also used in the SVD routines for reducing the rank of the matrix in solving the linear system of equations.
eta: Increment for the numerical calculation of the Jacobian.
fcmin: Minimum permitted value for the relaxation factor used in the damped Newton–Raphson algorithms.
sigma: Constant coefficient that determines the step with which the function $f = F.F/2$ decreases in the damped Newton–Raphson algorithms.
err1: Relative error in the integration algorithms.
err2: Absolute error in the integration algorithms.

METH_INT
meth0_int: Choose integration algorithm;
1 de – Adams–Bashforth method;
2 rkf7 – Runge–Kutta 7th order method depac package [28];
3 dderkf – Runge–Kutta-Fehlberg (4,5) method;
4 ddeabm – Adams–Bashforth method;
5 ddebdf – for stiff equations, backward differentiation formulae Numerical Recipes subroutines [6];
6 rkqs – Runge–Kutta 5th order;
7 bsstep – Bulirsch–Store method;
8 stifbs – Bulirsch–Store method for stiff equations.

METH_ROOT
meth0_root: Choose the zero finding method;
1 call other, a user-supplied algorithm;
2 call gnewt, damped Newton–Raphson algorithm
3 call secant, a secant-Broyden algorithm.

METH_LIN
meth0_lin: Choose a solver for the linear algebraic systems;
1 dlsvrr, singular value decomposition method;
2 dludcmp, lu-decomposition method;
3 dqrdcmp, qr-decomposition method.

METH_FUND
meth0_fund: meth0_fund is the same as the parameter -mshoot-;
1 compute the Jacobian (fundamental matrix) numerically;
2 compute the Jacobian analytically;
3 use odessa package [29] (not implemented) parameters in odessa package.
itask: 1 for normal computation of output values of y at t = tout.
itol: An indicator for the type of error control;
1 for scalar relative and absolute errors.
mf: Method flag.

iopt(1): 0 to indicate no optional inputs for integration.
iopt(2): 1 to indicate sensitivity analysis, = 0, to indicate no sensitivity analysis.
iopt(3): 0 no subroutine for inhomogeneity matrix df/dp.

ANALYSIS

iplot: Decide the rate of plotting the POs;
$+n$ every nth periodic orbits.
nppoints: The number of points stored to plot a PO.
npoinc: Decide to compute Poincare plane of sections;
$-n$ just bring the periodic orbit on the Poincare surface of section;
$+n$ the number of intersections of the trajectory with the Poincare surface of section.
meig: Decide the rate of computing the eigenvalues of the Monodromy Matrix of the located POs;
0 no eigenvalues;
$+n$ every nth periodic orbits.
ivect: Decide if you want to compute eigenvectors;
0 no eigenvectors;
$+n$ yes eigenvectors.
mmon: Choose how to calculate the monodromy matrix;
-1 use Monodromy Matrix as coming from the multiple shooting algorithm;
$+n$ the same as -mshoot-.
ifourie: Compute power spectra with FFT transforms;
0 no FFT;
$+n$ every nth trajectory.
ilyap: Compute the local maximum lyapunov exponent;
0 no MLE;
$+n$ every nth trajectory.
mcont: Choose variable for continuation, $1 \leq$ mcont $\leq$ ndiml.
msvar: Decide for constant or variable parameter continuation step;
0 constant step;
n variable step with n to be the optimum iteration number for convergence. Scale the parameter step by $n/$(No iterations of previous converged PO).
mextr: Decide for trivial or secant predictor in the continuation scheme;
0 trivial predictor, $x_{new} = x_{old}$;
1 secant predictor;
if (mcont.lt.0) extrapolate only the variable abs(mcont) at all nodes along the PO;
if (mcont.gt.0) extrapolate only variable mcont at $t = 0$;
if (mcont.eq.0) extrapolate all variables at time $t = 0$.

CCF

ntime: Decide to calculate the classical autocorrelation function;
0 no ccf;
$+n$ yes ccf.
ntrj: The number of trajectories used to compute ccf.
nv: The number of coordinates for which the ccf is computed.
iseed: Seed for the random number generator.
alpha0(1:ndiml): The widths of the Gaussians for the sampling of the trajectories.
enlim0: Upper bound for the energies of the selected trajectories.

WHAT Selecting *What* in the menu widget a window opens which allows you to load the help file po_t.what that explains the data.

4.3. Error diagnostics

The variable *kprint* notifies possible errors in cases of failure in converging to the appropriate PO.

For the damped Newton–Raphson algorithm, *kprint* means

kprint iter converged after -iter- iterations;
−1 converged to a stationary or turning point;
−2 NO convergence after nter iterations;
−3 NO convergence, stationary point in FMIN;
−4 possibly converged; need further tests.

For the secant algorithm, *kprint* means

kprint iter converged after -iter- iterations;
−1 converged to a stationary or turning point;
−2 NO convergence after nter iterations;
−3 NO convergence, stationary point in FMIN;
−4 NO convergence, singular matrix;
−5 NO convergence after trying reinitialization of the Jacobian.

5. Test run description

A Makefile creates the executable program. As a test case we use a relatively simple three-dimensional system with the Hamiltonian [30]

$$H = \tfrac{1}{2}(p_x^2 + p_y^2 + p_z^2) + \tfrac{1}{2}(\omega_x^2 x^2 + \omega_y^2 y^2 + \omega_z^2 z^2) - \epsilon x^2 y - \eta x^2 z\,. \tag{46}$$

The values of the parameters are $\omega_x^2 = 0.9$, $\omega_y^2 = 1.6$, $\omega_z^2 = 0.4$, $\epsilon = 0.08$, and $\eta = 0.01$. The harmonic frequencies satisfy the following resonance conditions; $\omega_x : \omega_y : \omega_z = 3 : 4 : 2$.

The subroutines POTEN and SECDER (given in Appendix A) are stored in the directory *applications*. Here is the content of the output file po_t.out for one PO.

```
*******************************************************************************
*******************************************************************************
                              PO-Search-for-X2YZ
*******************************************************************************
*******************************************************************************
No of equations = 6
No of nodes = 4
method of integration = 1
method of root search = 2
method of solving l.eq. = 2
read initial vector = 0
fixing one variable = 0
const. or var. jacobian = 1
Equilibrium point
.00000000 .00000000 .00000000 .00000000 .00000000 .00000000
Angular variables : 0
Adiabatic variables : 0
```

integr. time step = .10000000E+00
initial period = .66253800E+01
variabl. for PSS (cont.)= 2
max. iter. in newton = 50
No of located p.o. = 1
increment step for p.o. = .10000000E-02
errf, errx = .10000000E-07 .10000000E-07
tolerance for ill-cond. = .10000000E-07
odessa v. itask,itol,mf = 1 1 11
iopt = 0 1 0
differentiation const. = .10000000E-06
minimum relaxation fac. = .10000000E-01
sigma = .10000000E+00
abserr, relerr in int. = .10000000E-11 .10000000E-11
direction in time = 1
direction in integr. = 1
iplot = 0
nppoints = 500
npoinc = 0
meig = 1
ivect = 0
mshoot = 2
ifourie = 0
ilyap = 0
mcont = 7
msvar = 0
mextr = 0
ntime = 0
ntrj = 1
nv = 1
iseed = 123456789
****
widths of gaussians
8.00000000 8.00000000 8.00000000 8.00000000 8.00000000 8.00000000 .00000
****
Starting point in phase space
.00000000 .01000000 .00336900 .46408150 .00000000 .00000000
6.62538000
****
analytic jacobian seems to be o.k.
****
****
TRAJ. No = 1
****
X(1) = .000078421587 X(2) = .010786284687 X(3) = .003369013419
X(4) = .464085471270 X(5) = -.000002915863 X(6) = -.000000227704
Period = 6.625380000000
Energy = .107783010288
Accur. = .92E-08
Kprint = 1
Action = .113633177667

Eigenvalues

real	imaginary	norm	phase (in degrees)
-.499361993394D+00	.867405522180D+00	.100087698563D+01	.119928867251D+03
-.499361993394D+00	-.867405522180D+00	.100087698563D+01	-.119928867251D+03
-.498487277651D+00	.865886116868D+00	.999123782804D+00	.119928867251D+03
-.498487277651D+00	-.865886116868D+00	.999123782804D+00	-.119928867251D+03
.999999999917D+00	.128911023608D-04	.100000000000D+01	.738605758566D-03
.999999999917D+00	-.128911023608D-04	.100000000000D+01	-.738605758566D-03

Determinant = .1000000000E+01

Denper = .00000000E+00

*** Job ended with kprint = 1

Appendix A

```
      subroutine poten(t,y,f,nn)
      implicit real*8(a-h,o-z)
      include '../parameter.inc'
      dimension y(nde),f(nde)
      dimension jdiab(ndiml/2)
      common/scale/iperiod_scale
      common/adiab/idiab,jdiab
      common/netr/period,deltat,netrial
      common/energy/etot,vpot
********* YOU CAN MODIFY AFTER THIS ENTRY ******************************
      DATA A,B,C,EPSIL,ETA/0.9D0,1.6D0,0.4D0,0.08D0,0.01D0/
      vpot=0.5d0*(a*y(1)*y(1)+b*y(2)*y(2)+c*y(3)*y(3))
     +     -y(1)*y(1)*(epsil*y(2)+eta*y(3))
      f(1)=y(4)
      f(2)=y(5)
      f(3)=y(6)
      f(4)=-(a*y(1) - 2.0d0*y(1)*(epsil*y(2)+eta*y(3)))
      f(5)=-(b*y(2) - epsil*y(1)*y(1))
      f(6)=-(c*y(3) -   eta*y(1)*y(1))
      etot = 0.5d0*(y(4)*y(4) + y(5)*y(5) + y(6)*y(6)) + vpot
********* DO NOT MODIFY AFTER THIS ENTRY ******************************
      if(iperiod_scale.ne.0) then
         do i=1,ndiml
            f(i)=f(i)*period
         enddo
      endif
      if(idiab.ne.0) then
         kdiab=abs(idiab)
         do i=1,kdiab
            f(jdiab(i))=0.0d0
            f(jdiab(i)+ndiml/2)=0.0d0
         enddo
      endif
      f(ndiml+1)=0.0d0
      return
      end
      subroutine secder(t,y,a,nn)
      implicit real*8(a-h,o-z)
      include '../parameter.inc'
      double precision t,y,a
      dimension y(nde),a(nde,nde)
```

```
      dimension jdiab(ndiml/2)
      common/energy/etot,vpot
      common/netr/period,deltat,netrial
      common/scale/iperiod_scale
      common/adiab/idiab,jdiab
********* YOU CAN MODIFY AFTER THIS ENTRY ******************************
      DATA A1,B,C,EPSIL,ETA/0.9D0,1.6D0,0.4D0,0.08D0,0.01D0/
      do i=1,6
         do j=1,6
            a(i,j) =0.0d0
         enddo
      enddo
      a(1,4)= 1.0d0
      a(2,5)= 1.0d0
      a(3,6)= 1.0d0
      a(4,1)=-(a1 - 2.0d0*(epsil*y(2)+eta*y(3)))
      a(4,2)=-(- 2.0d0*y(1)*epsil)
      a(4,3)=-(- 2.0d0*y(1)*eta)
      a(5,1)=-( -2.0d0*epsil*y(1))
      a(5,2)=-b
      a(6,1)=-( -2.0d0*eta*y(1))
      a(6,3)=-c
********* DO NOT MODIFY AFTER THIS ENTRY ******************************
      if(iperiod_scale.ne.0) then
          do i=1,ndiml
            do j=1,ndiml
               a(i,j)=a(i,j)*period
            enddo
         enddo
      endif
      if(idiab.ne.0) then
         kdiab=abs(idiab)
         do i=1,kdiab
            do j=1,ndiml
               a(jdiab(i),j)=0.0d0
               a(jdiab(i)+ndiml/2,j)=0.0d0
               a(j,jdiab(i))=0.0d0
               a(j,jdiab(i)+ndiml/2)=0.0d0
            enddo
         enddo
      endif
      do i=1,ndiml+1
         a(ndiml+1,i)=0.0d0
         a(i,ndiml+1)=0.0d0
      enddo
      return
      end
```

References

[1] S.C. Farantos, Int. Rev. Phys. Chem 15 (1996) 345–374.
[2] S.C. Farantos, H.-M. Keller, R. Schinke, K. Yamashita, K. Morokuma, J. Chem. Phys. 104 (1996) 10055.
[3] C. Beck, H.-M. Keller, S.Y. Grebenshchikov, R. Schinke, S.C. Farantos, K. Yamashita, K. Morokuma, J. Chem. Phys. (1997), in press.
[4] P. Gaspard, I. Burghardt, Adv. Chem. Phys. 101 (1997) 491.
[5] S.C. Farantos, Chem. Phys. 159 (1992) 329.

[6] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes (Cambridge Univ. Press, Cambridge, 1986).
[7] L. Zhang, Appl. Math. & Comput. 42 (1991) 209.
[8] L. Zhang, Numer. Math. 66 (1993) 399.
[9] U. Feudel, W. Jansen, Int. J. Bifurc. & Chaos 2 (1992) 773.
[10] M.A.M. Aguiar, C.P. Malta, M. Baranger, K.T.R. Davies, Ann. Phys. 180 (1987) 167.
[11] M. Baranger, K.T.R. Davies, J.H. Mahoney, Ann. Phys. 186 (1988) 110.
[12] K.T.R. Davies, T.E. Huston, M. Baranger, Chaos 2 (1992) 215.
[13] E.J. Doedel, Cong. Numer. 30 (1981) 235.
[14] E.J. Doedel, X.J. Wang, AUTO94, software for continuation and bifurcation problems in ordinary differential equations, Technical report, Center for Research on Parallel Computing, California Institute of Technology, Pasadena CA 91125 CRPC-95-2 (1995).
[15] P. Deuflhard, Numer. Math. 33 (1979) 115.
[16] P. Deuflhard, Numer. Math. 22 (1974) 189.
[17] H.B. Keller, Numerical Solution of Two Point Boundary Value Problems, Vol. 24, Society for Industrial and Applied Mathematics (1976).
[18] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis (Springer, New York, 1980).
[19] R. Seydel, From Equilibrium to Chaos: Practical Bifurcation and Stability Analysis (Elsevier, Amsterdam, 1988).
[20] E. Reithmeier, Periodic Solutions of Nonlinear Dynamical Systems, Lecture Notes in Mathematics (Springer, Berlin, 1991).
[21] A.J. Lichtenberg, M.A. Libermann, Regular and Stochastic Motion (Springer, New York, 1981).
[22] J.M.G. Llorente, E. Pollak, Ann. Rev. Phys. Chem. 43 (1992) 91.
[23] M. Henon, Physica D 5 (1982) 412.
[24] S.C. Farantos, in: Time Dependent Quantum Mechanics: Experiments and Theory, J. Broeckhove, ed. (Plenum Press, New York, 1992).
[25] R. Seydel, Int. J. Bifurc. & Chaos 1 (1991) 3.
[26] J.K. Ousterhout, Tcl and the TK Toolkit (Addison-Wesley, Reading, MA, 1994).
[27] B.B. Welch, Practical Programming in Tcl and Tk (Prentice Hall, Englewoods Cliffs, NJ, 1995).
[28] L.F. Shampine, M.K. Gordon, Computer Solution of Ordinary Differential Equations: The Initial Value Problem (Freeman, San Francisco, CA, 1975).
[29] J.R. Leis, M.A. Kramer, ACM Trans. Math. Software (1985).
[30] G. Contopoulos, S.C. Farantos, H. Papadaki, C. Polymilis, Phys. Rev. E 50 (1994) 4399.