A semi-classical approach for Lyapunov exponents of a quantum mechanical system

Fernando Pérez. University of Colorado, Boulder.

19th April 2002

Abstract

We attempt to construct Lyapunov exponents for quantum mechanical systems. Lyapunov exponents have always been a key concept of classical chaotic dynamics, but difficult to use in a quantum context. By constructing a semi-classical potential from the ground state wavefunction of a system and analyzing the classical dynamics produced by this potential, one can use concepts from classical dynamics, such as Lyapunov exponents, to probe the dynamics of the quantum system. In this manner, we find non trivial corrections to the classical dynamics arising from quantum information: the chaotic behavior is suppressed although not entirely eliminated.

1 Introduction

The concept of Lyapunov exponents, which characterize the phase-space divergence of infinitesimally neighboring trajectories for dynamical systems, has long been a useful criterion for the identification of chaotic systems. However, since by construction they rely on the idea of a point trajectory in phase space, their application to quantum mechanical problems has proved elusive. The domain of "quantum chaos" is a murky one, even the very definition of the term not being clear, and the problem of not having the analog of classical trajectories is at the heart of the matter. For an introductory review which analyzes a classical example, the kicked rotor, see [1]. Gutzwiller [2] provides a comprehensive study of the problem, with many of his original contributions to the field.

Quantum mechanics describes the world using a radically different language from that of classical mechanics. For problems with time-independent potentials—the case we consider here—the dynamics is trivially integrated out and the only remaining question of interest is the eigenmode structure of the potential and associated boundary conditions. It then appears as if the wave nature of quantum systems and the resulting interference phenomena might "wash out" the rich classical dynamics, leaving only a smooth landscape of wavefunctions.

The obvious question which now arises is that posed by the correspondence principle: in the limit of high quantum numbers, where we expect the quantum behavior to somehow smoothly blend into the classical one, how can this occur? For classically integrable systems the WKB idea provides some insight, but these are precisely the cases where the classical dynamics is simple, ultimately reducible to oscillatory motions in many dimensions (via action-angle variables). However, classically chaotic systems appear untractable by this approach: a knowledge of *all* periodic orbits is needed, and for a chaotic system these form a dense set in phase space. Great progress has been made in the use of so called trace formulas [2], but we still lack a true understanding of the connection between classical chaotic dynamics and quantum mechanics.

One avenue of attack to the problem, inspired in ideas originally proposed by D. Bohm in 1952 [3], considers a the wavefunction of a quantum system as a field which physically drives the dynamics of classical point particles. Much work has been devoted in recent years to this approach ([4]-[10]), from proving rigorously that all of the usual results of "traditional" quantum mechanics can be recovered by interpreting quantities appropriately, to using the resulting classical dynamics to define quantum Lyapunov exponents and probe the classical-quantum correspondence in new ways.

This paper follows an idea of similar nature, but different in the particulars. We will also construct a classical potential from the quantum problem and then probe the resulting "mixed" or semi-classical dynamics, study the evolution of orbits and compare to the purely classical problem. However, the semi-classical potential will be obtained using a prescription for a supersymmetry improvement to WKB methods.

We will use the method to analyze a nonlinear oscillator described by the potential $V(x,y) = \frac{1}{2}(x^2+y^2) + 4kx^2y^2$, whose classical dynamics (chaotic for $k \neq 0$) will be studied and compared with the resulting semi-classical dynamics.

2 The semi-classical potential

Consider a two dimensional quantum particle of mass m in a potential $V(x,y)$, with ground state wavefunction $\psi(x,y)$. Without loss of generality we can write

$$\psi(x,y) = N\exp(-W(x,y)/\hbar) \tag{1}$$

where N is a normalization constant. Using Schrödinger's equation

$$\left[-\frac{\hbar}{2m}\nabla^2 + V(x,y)\right]\psi(x,y) = E\psi(x,y) \tag{2}$$

we can then obtain the following formal expression (from now on we let $m=1$)

$$U(x,y) \equiv V(x,y) - E = \frac{\hbar}{2}\frac{\nabla^2\psi(x,y)}{\psi(x,y)} \tag{3}$$

It is customary in the literature to define

$$\Phi(x,y) \equiv \frac{1}{\sqrt{2}}\nabla W(x,y) \tag{4}$$

which using (1) allows us to write $U(x,y)$ as

$$U(x,y) = \Phi^2(x,y) - \frac{\hbar}{\sqrt{2}} \nabla \cdot \Phi(x,y) \tag{5}$$

or equivalently

$$U(x,y) = U_{tree}(x,y) + U_{loop}(x,y) \tag{6}$$

with the obvious definitions

$$\begin{aligned} U_{tree}(x,y) &\equiv \Phi^2(x,y) \\ U_{loop}(x,y) &\equiv -\tfrac{\hbar}{\sqrt{2}} \nabla \cdot \Phi(x,y) \end{aligned} \tag{7}$$

The designations “tree” and “loop” for the two parts that make up $U(x,y)$ come from a supersymmetry inspired analysis which we will not cover here. See [11] for a detailed discussion of the applications of supersymmetry to classical and quantum mechanics. The essential fact which we will use here is a realization made in 1985 by Comtet, Bandrauk and Campbell [12] that using $U_{tree}(x,y)$ one could construct a WKB-type formula which gives far better results than the traditional one, and which for a class of potentials known as *shape invariant* provides an exact solution.

This SUSY WKB formula (for a one-dimensional system), is to lowest order in $\hbar$ given by

$$\int_a^b \mathrm{d}x \sqrt{2m\left[E_n - \Phi^2(x)\right]} = \int_a^b \mathrm{d}x \sqrt{2m\left[E_n - U_{tree}(x)\right]} = n\pi\hbar, \tag{8}$$

where a and b are the classical turning turning points for $U_{tree}(x)$: $E_n = U_{tree}(a) = U_{tree}(b)$. The remarkable successes of this formula suggest that somehow the potential U_{tree} “knows” about the structure of the quantum poblem, and can successfully use this information in a semiclassical context. This led us to attempt using it to study the connection between the quantum and classical regimes for a chaotic system.

U_{tree} is explicitly given by

$$U_{tree}(x,y) = \frac{\hbar^2}{2} \left[\frac{\nabla\psi(x,y)}{\psi(x,y)} \right]^2 \tag{9}$$

and it is interesting to note that for the case when our system of study is regular, $k = 0$, one can easily show (since the ground state exact wavefunction is readily available) that $U_{tree}(x,y) = V(x,y)$. However as we will later see, for the classically chaotic cases ($k \neq 0$) this is not the case and interesting differences appear.

3 Numerical procedure

Our roadmap is to first solve the quantum problem for a given value of k, computing the ground state wavefunction $\psi(x,y)$. Using this we then construct the semiclassical potential $U_{tree}(x,y)$ and use it to solve the classical dynamics.

3.1 The quantum problem

To solve the time independent Schrödinger equation we used the Finite Element Method (FEM) [13]. This is a variational type method which uses a *local* basis instead of the single global function used in traditional variational calculations.

Even though the idea of the method is very simple to state, the programming can become quite involved because of bookkeeping issues. We will not go into details here, simply stating the basic philosophy behind the FEM: we break the region of interest into many small elements where we perform a variational solution using fairly simple trial functions. The global solution is then constructed by matching all the local solutions across element boundaries. The elements at the outside boundary of the region account for the global boundary conditions of the problem. The variational solution for the local functions leads to the solution of a generalized eigenvalue problem, which can be done with many of the available libraries of numerical routines. Our program uses routines from Numerical Recipes [14], but any other standard library such as EISPACK would equally work.

This approach allows great flexibility since one can arbitrarily improve—within the obvious time and memory constraints—the quality of the solution by adding more elements as needed. If the problem is such that significantly different length scales need to be considered in different regions, the method can be adapted by distributing the element grid in a non uniform way, with denser coverage in the areas with short relevant length scales.

Another important property of the FEM is that by choosing sufficiently smooth local trial functions one can construct a global solution with any necessary degree of smoothness: continuity in the solution, its first or even its second derivatives (as long as such derivatives exist in principle, from a mathematical standpoint). This is very important for our problem because $U_{tree}(x, y)$ needs $\nabla\psi(x, y)$. In this project we have used a grid with 9 elements and 3 nodes per element, and Hermite interpolation polynomials[1] which guarantee continuity in the first derivative of the solution.

In summary, the FEM is capable of providing very accurate solutions but is fairly expensive computationally: the size N of the matrices involved is $N \sim (N_{nodes})^2$, where N_{nodes} is the number of nodes per side on the grid. Solving an $N \times N$generalized eigenvalue problem for large N can be quite a daunting task, and obviously the memory requirements become quickly prohibitive: $Mem \sim 384\,(N_{nodes})^4$ Mb! The 384 factor is for a solution with continuous first derivative, in double precision.

3.2 The semi-classical problem

A fifth order adaptive Runge-Kutta integrator [14] was used to solve the classical equations of motion both for the classical potential $V(x, y)$ and the semi-classical one $U_{tree}(x, y)$. See the Appendix for part of this code. At every point (x, y) of

[1]These should not be confused with the Hermite polynomials $H_n(x)$ used in the analytical solution of the quantum simple harmonic oscillator. See [13] for details.

integration of the equations of motion, the program calculates the force acting on the particle using the two-point centered formula

$$F_x(x,y) = -\frac{U_{tree}(x+h,y) - U_{tree}(x-h,y)}{2h}, \tag{10}$$

with a similar expression for $F_y(x,y)$.

Two initially nearby orbits were computed and their separation in phase space was tracked, defined as

$$\delta(t) \equiv \sqrt{[x_2(t)-x_1(t)]^2 + [y_2(t)-y_1(t)]^2 + [v_{x2}(t)-v_{x1}(t)]^2 + [v_{y2}(t)-v_{y1}(t)]^2} \tag{11}$$

The Lyapunov exponent for a given pair of initially nearby trajectories can then be estimated by making the assumption that

$$\delta(t) \sim e^{\lambda t}. \tag{12}$$

A positive value for λ signals chaotic behavior, since trajectories which start arbitrarily close will diverge exponentially fast.

For the calculation of the Poincaré sections a separate program was used (which had been previously developed). It integrates only the classical equations, but using an extremely accurate method which exploits the symmetries of $V(x,y)$ to produce a very high order method capable of long time integrations with minimal error. Details of this will not be given here as this was previous work not done during this course, a description of the algorithm can be found in [15]. The Poincaré sections are computed by detecting when the trajectory has crossed the surface and reverse integrating along the orbit (after a suitable reparametrization) in order to avoid all interpolation errors [16].

4 Analysis of the system as k is varied

We will now present results for our 2-d non-linear oscillator with potential

$$V(x,y) = \frac{1}{2}(x^2+y^2) + 4kx^2y^2$$

as we vary k from $k=0$ where the system is regular to $k=4$ where full chaos has developed.

A plot of $V(x,y)$ for $k=2$ is shown in Fig. 1, with contours which clearly show the highly non-linear nature of the potential.

All orbits shown here for different values of k were generated with initial conditions $(x_0,y_0)=(1,0)$, $(v_{x0},v_{y0})=(0,1)$, which gives the system an total energy $E=1$. Since this is the ground state energy (in units of $\hbar$) for the quantum system (at $k=0$) it was taken as a reference energy suitable for the whole study. To compute the separation of orbits we let evolve a second initial condition with $x_0^{'} = x_0 + \epsilon,\ \epsilon = 10^{-5}$ and all other values equal.

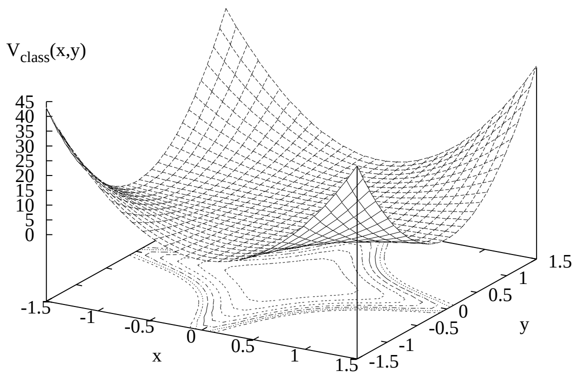

Figure 1: Classical potential $V(x, y)$ for $k = 2$.

4.1 The linear case $(k = 0)$

The trajectories in the linear case are as we know just circles, both classical and semi-classical, they are presented in Fig. 2. As we mentioned before, for $k = 0$ one can show analytically that $U_{tree}(x, y) = V(x, y)$ so it is no surprise that both the classical and semi-classical trajectories come out identical. It is, however, a nice check for the quality of our numerics.

The phase space separation $\delta(t)$ is plotted in Fig. 3: we see that the classical orbit is perfectly regular, and for the semiclassical one the small growth of $\delta(t)$ simply signals numerical noise which is more significant (as expected) for the semiclassical case. One should remember that $\delta(t)$ is the phase space separation for two initially very close trajectories for *each* case (classical or semiclassical), it isn't a comparison between classical and semiclassical. That comparison is made by looking at both $\delta_{class}(t)$ and $\delta_{semiclass}(t)$, since they are in general computed for different potentials ($U_{tree}(x, y) \neq V(x, y)$ for all $k \neq 0$).

4.2 Non-linear behavior at $k = 0.2$

For $k = 0.2$, two nearby classical orbits are shown in Fig. 4. In Fig. 5 we plot both the classical and semi-classical orbits starting from the *same* initial condition, and already the effects of the different potentials are visible.

The divergence of the trajectories for both potentials is shown in Fig. 6. We start here to see the interesting phenomenon, which has been previously reported using different methods of analysis, of quantum suppression of classical chaos. While for the classical case we find a Lyapunov exponent $\lambda_c \approx 0.15$, for the semiclassical one we see $\lambda_{sc} \approx 0.05$[2]. Since this is a logarithmic quantity, such

[2]These values have been estimated by an "eye-fit" to $\delta(t)$ in the region before its value

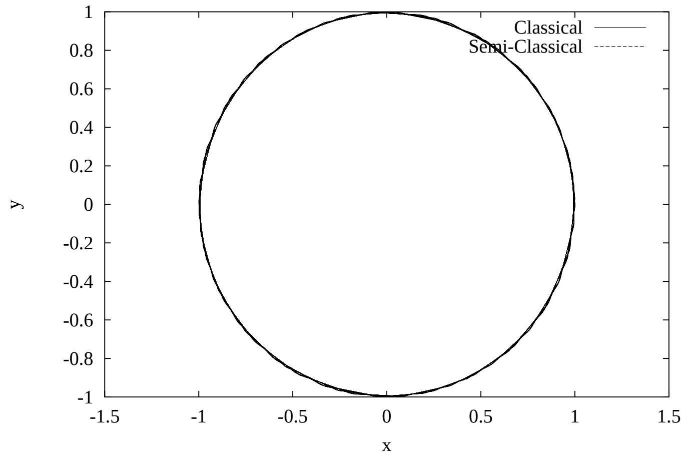

Figure 2: Classical and semi-classical trajectories for $k = 0$.

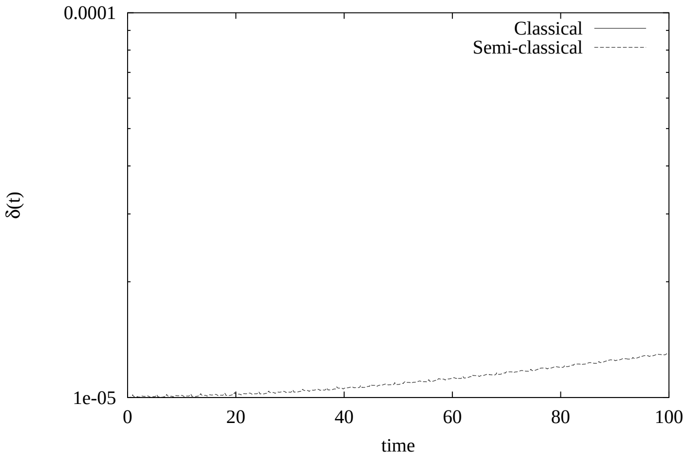

Figure 3: Phase space separation $\delta(t)$ between two initially very close trajectories for both the classical and semi-classical cases.

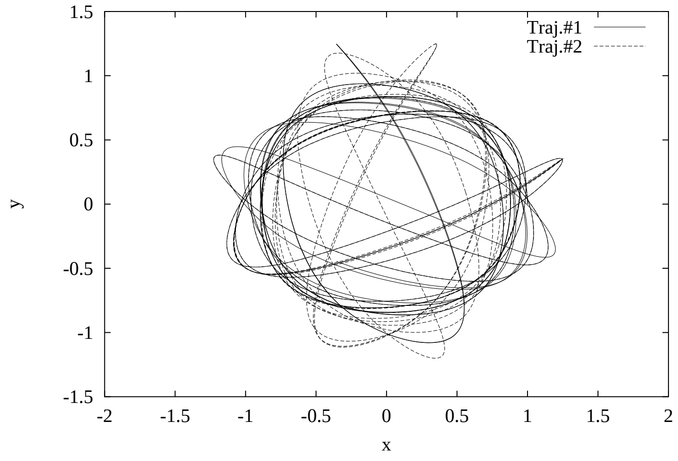

Figure 4: Evolution of two nearby classical orbits for $k = 0.2$.

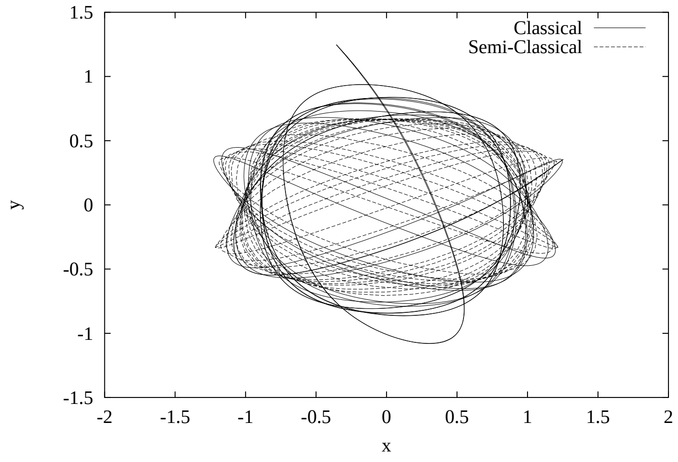

Figure 5: Comparison of the classical and semi-classical orbits for $k = 0.2$.

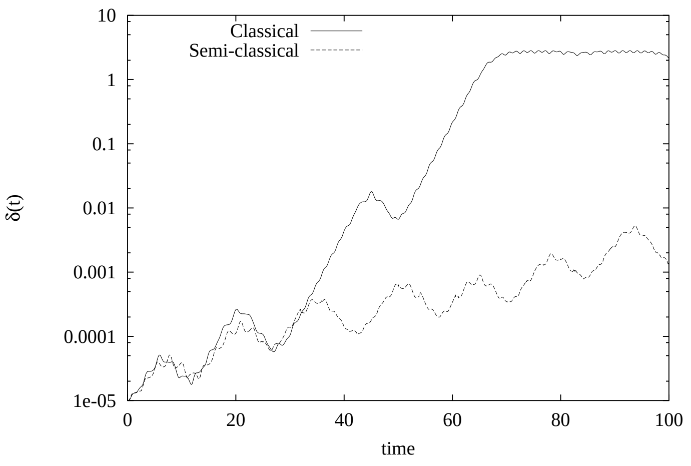

Figure 6: $\delta(t)$ for both classical and semi-classical orbits at $k = 0.2$.

a difference is significant even with our crude estimations.

A look at the classical Poincaré section for $k = 0.2$ in Fig. 7 shows that the system already has regions of chaotic behavior, even though regular ones still persist. This is exactly the kind of transition to chaos scenario predicted by the celebrated KAM theorem [17].

4.3 Behavior at $k = 1$

The non-linear effects are obviously stronger for $k = 1$. The classical orbits are shown in Fig. 8 and the "quantum" (semi-classical) ones in Fig. 9. By now the difference between the two scenarios is very marked, with the classical behavior having become basically chaotic but the semiclassical one still very regular.

If we superimpose one classical with one semi-classical orbit (Fig. 10) this difference is very apparent: while the classical trajectory explores most of the available configuration space, the semiclassical one remains confined to a smaller region and evolves in a much more regular manner.

The phase space distance $\delta(t)$ shows also these differences (Fig. 11), and we estimate for the Lyapunov exponents the values $\lambda_c \approx 0.45$ and $\lambda_{sc} \approx 0.05$.

This behavior comes from the marked difference between $V_{class}(x, y)$ and $U_{tree}(x, y)$, as seen in Figs. 12 and 13. The contours at the base of the plots show clearly that $U_{tree}(x, y)$ is a more linear potential than $V_{class}(x, y)$, even though both correspond to the same k value.

saturates, so they must only be taken as rough estimates and not as numerically accurate. What is of interest to us here is the clear difference between the classical and semi-classical behavior, already apparent from the $\delta(t)$ plots.

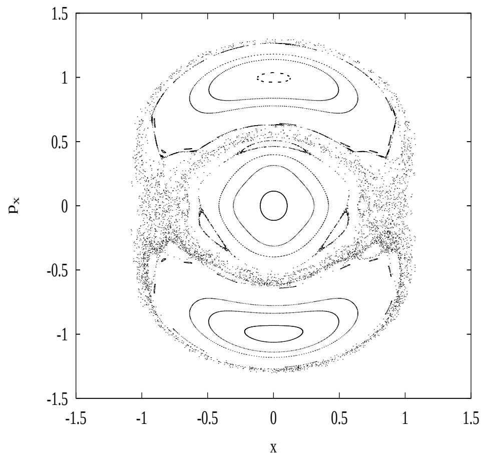

Figure 7: Poincaré section for the classical potential $V(x,y)$ at $k = 0.2$.

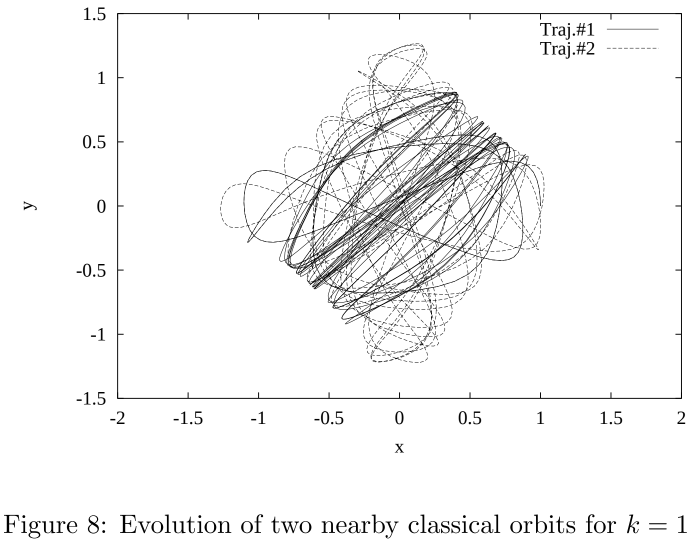

Figure 8: Evolution of two nearby classical orbits for $k = 1$.

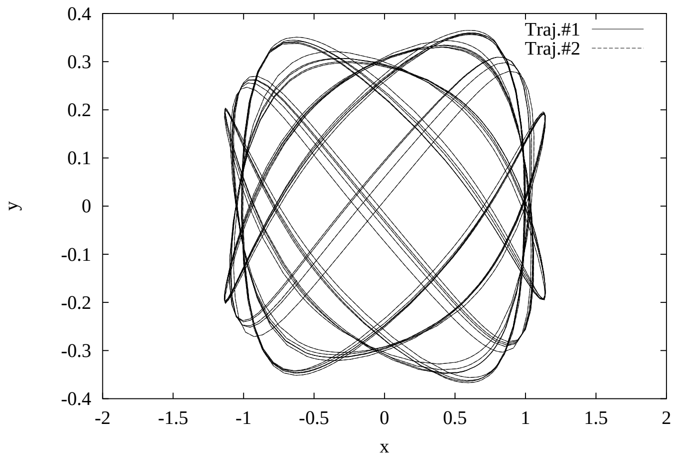

Figure 9: Evolution of two nearby semi-classical orbits for $k = 1$.

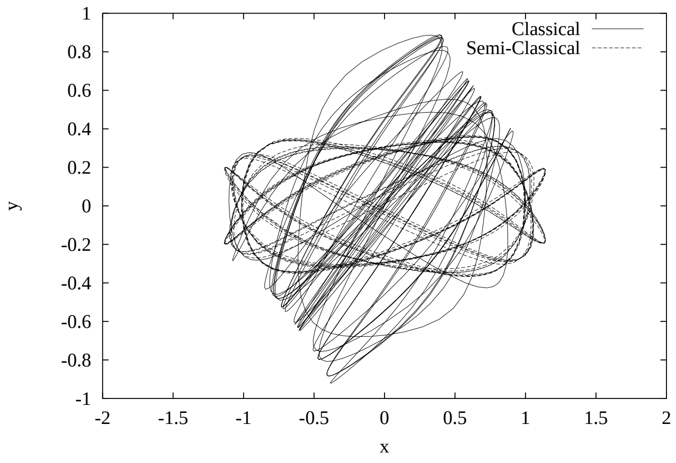

Figure 10: Comparison of the classical and semi-classical orbits for $k = 1$.

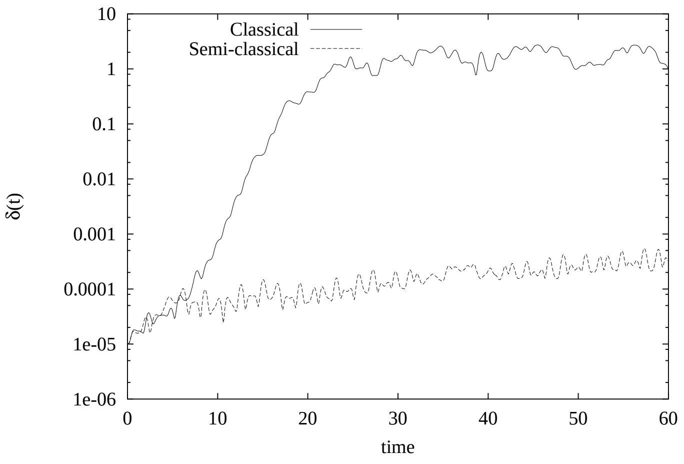

Figure 11: $\delta(t)$ for both classical and semi-classical orbits at $k = 1$.

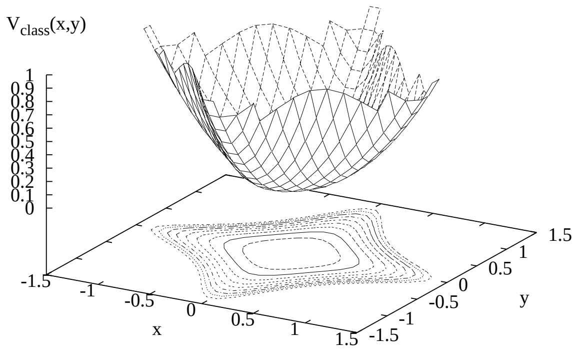

Figure 12: Classical potential $V_{class}(x, y)$ for $k = 1$.

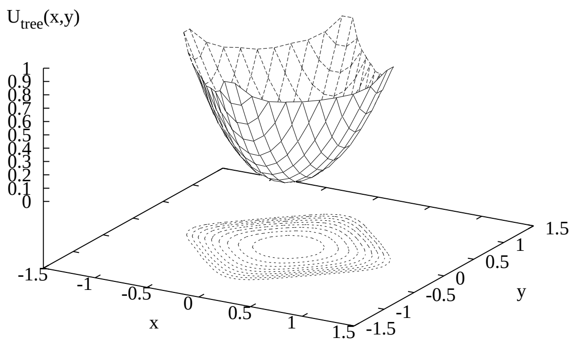

Figure 13: Semi-classical potential $U_{tree}(x, y)$ for $k = 1$.

4.4 Fully developed chaos at $k = 4$

When we reach $k = 4$ both systems have developed chaotic behavior, even though differences remain. Classical and quantum orbits are plotted in Figs. 14 and 15 (two initially nearby orbits per plot) and the chaotic motion is apparent in both cases. The orbits cover essentially all of the available configuration space, as can be seen by comparing the shape of the resulting dark region with the contours for both potentials (Figs. 17 and 18).

However, the quantum system still shows suppression of chaos, as evidenced by a plot of $\delta(t)$ in Fig. 16, from which we estimate the values $\lambda_c \approx 1.2$ and $\lambda_{sc} \approx 0.4$. Both systems have become chaotic, but the divergence of nearby orbits is slower for the semi-classical one.

The difference in the potentials is quite clear, and we show plots of both using two different vertical scales: in Figs. 17 and 18 the maximum value plotted is 1, so we are only seeing the region actually available to our test particles (since they both start with energy $E = 1$ and the system is conservative). In Figs 19 and 20 we plot up to $E = 10$ to show the full shape of the potentials.

The fact that our system is fully chaotic at $k = 4$ is readily apparent from the Poincaré surface of section shown in 21: a single trajectory covers the entire available phase space, meaning that no torus structure remains. The only integral of the motion is now the energy E and the behavior is essentially ergodic.

5 A global look at orbit divergence.

We finally make a comparison between the classical and semi-classical cases by plotting $\delta(t)$ for the several values of k analyzed on the same plot (Figs. 22 and

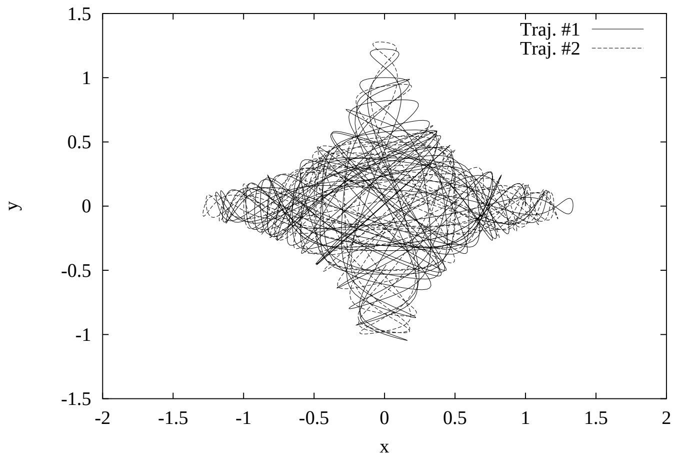

Figure 14: Evolution of two nearby classical orbits for $k = 4$.

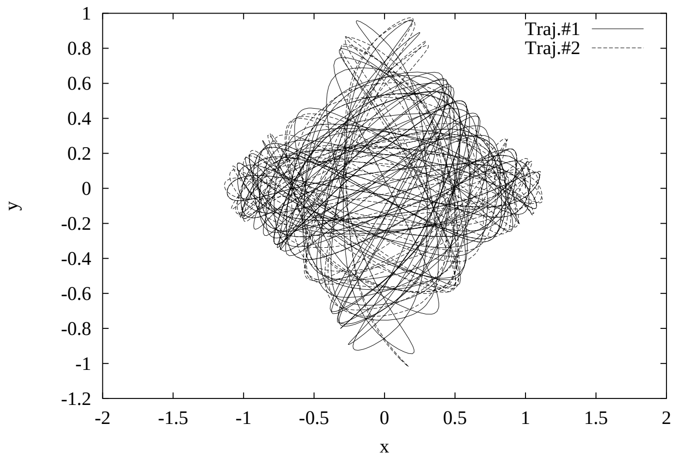

Figure 15: Evolution of two nearby semi-classical orbits for $k = 4$.

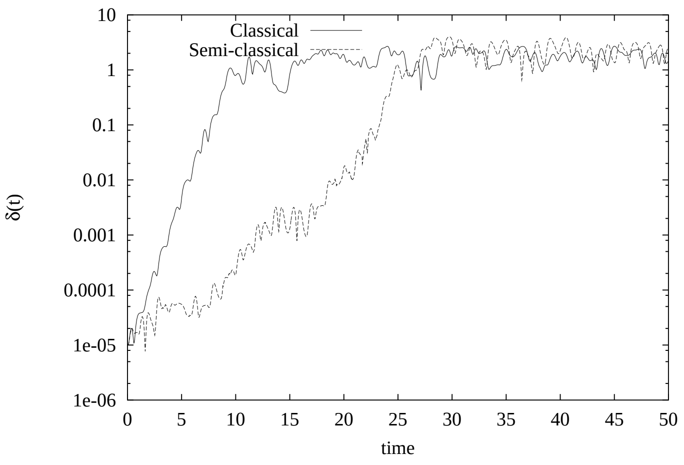

Figure 16: $\delta(t)$ for both classical and semi-classical orbits at $k = 4$.

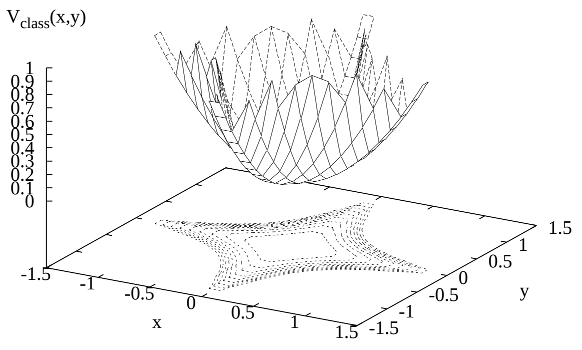

Figure 17: Classical potential $V_{class}(x, y)$ for $k = 4$.

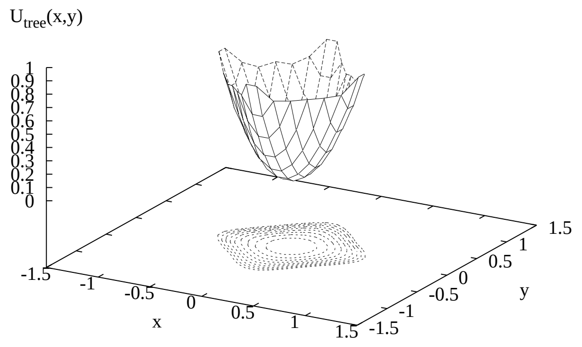

Figure 18: Semi-classical potential $U_{tree}(x, y)$ for $k = 4$.

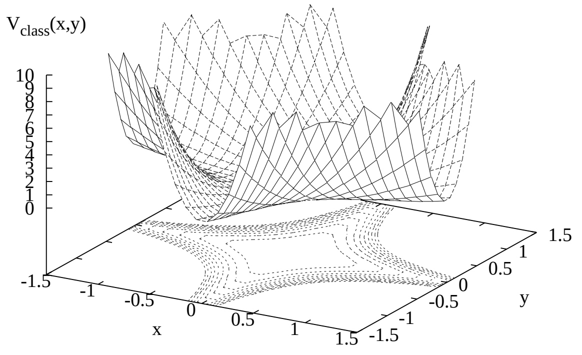

Figure 19: Classical potential $V_{class}(x, y)$ for $k = 4$, using a different vertical scale from that of Fig. 17.

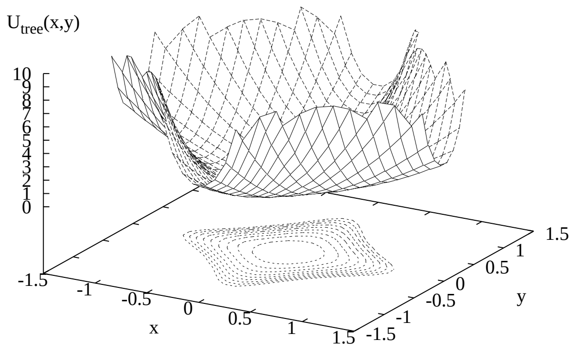

Figure 20: Semi-classical potential $U_{tree}(x, y)$ for $k = 4$, using a different vertical scale from that of Fig. 18.

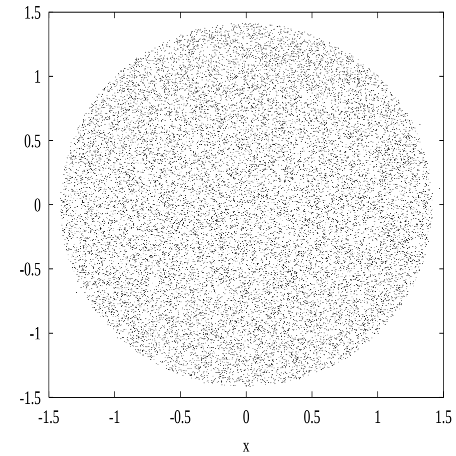

Figure 21: Poincaré section for the classical potential $V(x, y)$ at $k = 4$.

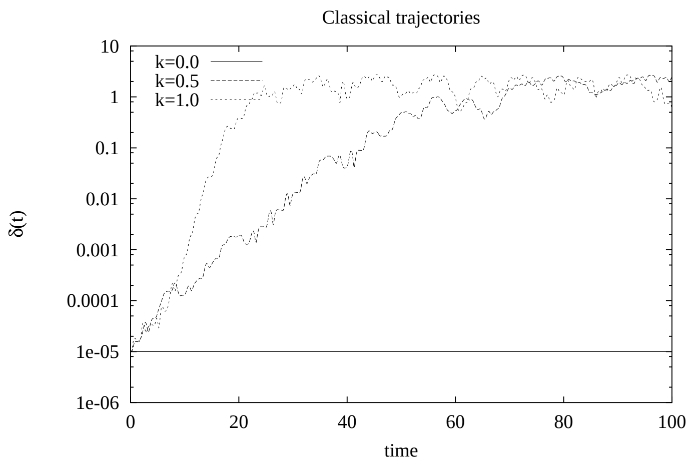

Figure 22: $\delta(t)$ for various values of k for the classical potential.

23).

It is clear how the classical system makes the transition to chaos fairly rapidly, with only the $k = 0$ case being truly regular. In contrast, the semi-classical system remains reasonably regular for all values of k up to $k = 4$ (since $\delta(t)$ is shown on a logarithmic scale the differences are very significant).

6 Conclusions and Outlook

- We have found the potential $U_{tree}(x, y)$ to provide non-trivial quantum corrections to the classical behavior. A quantum suppression of chaos is observed but not a complete elimination of the phenomenon, since eventually even the semi-classical system shows chaotic behavior.

- Our analysis has an important limitation: we have taken a look at single trajectories for all the comparisons, and a more sensitive probe would be to construct Poincaré sections for $U_{tree}(x, y)$. This would be useful to clarify how phase space is globally affected by the quantum effects. It was not done as part of this project because of time constraints but can be easily done in the near future, since it requires only straightforward extensions to the existing numerical tools.

- A 2-*d* SUSY WKB? It would be very interesting to make a 2-*d* extension of SWKB, since we have seen how the quantum potential $U_{tree}(x, y)$ carries non-trivial information from the quantum to the classical regime. However, this is immensely difficult in fully chaotic regimes because one needs *all* periodic orbits of the system, but for a chaotic system these

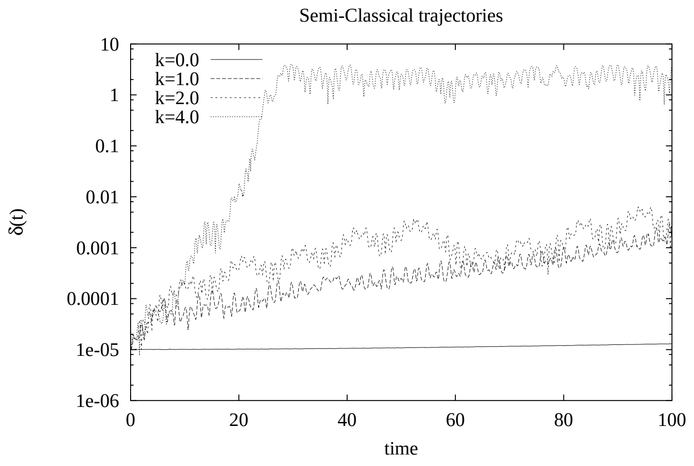

Figure 23: $\delta(t)$ for various values of k for the semi-classical potential.

orbits are dense in phase space. This is truly the heart of the difficulty of understanding how a classically chaotic system behaves in the quantum regime, and it is interesting to mention that as far back as 1917 Einstein already had a glimpse of the formidable problems lying here.

- However, *maybe* for small k the quantum suppression of chaos is enough to allow the WKB approach... This is just an idea mentioned as a possibility for future exploration, since it might allow to probe the quantum non-perturbative regime in a novel way.

Acknowledgements

I am very grateful to Jorge Mahecha (Universidad de Antioquia, Medellín, Colombia) for permanent guidance and fruitful discussions on this problem, as well as for the finite element code used to solve the time independent Schrödinger equation.

Appendix

We will show here only some of the numerical code used in this project. Three main separate programs (plus various control and data processing scripts) were written:

- One to solve the classical dynamics to very high precision using a special integration technique (not discussed here). This was the program used to compute the Poincaré surfaces of section shown. Done in C++.

- The finite element solver for the quantum problem. Done in Fortran.
- A general integrator capable of solving classical dynamics using either the classical potential or the quantum one, which it constructs from the finite element solution of the ground state wavefunction. Done in C++.

To avoid extending this paper excessively and to present only the work done specifically over the last two months for this project, we will then include only the code for the third program. The following is the file with the main integration and numerical routines, other utility files are not shown.

```
/* ****************************************************************************
   Quantum-classical oscillator
   ============================
   Program to compute classical trajectories and Lyapunov exponents for a
   particle in a potential obtained by numerical integration of a 2-d
   Schrodinger equation.

   Fernando Perez, University of Colorado, Dec/98.

   Parts taken from previous Fortran program by Prof. Jorge Mahecha G.
   Translated to C++ with f2c and manual editing.

****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <string.h>
#include "nr_summ.h"
#define NRANSI
#include "nrutil.h"
#include "utils.h"

// do everything in double, brute force way
#define float double

//***************************************************************************
/* Common Block Declarations */
struct {
  double ancho, xmaxi, dista, emass, kq_nlin;
} param;

struct {
  double zn[10], a[1600], b[1600];    // were [40][40] in Fortran code
  long int indx[40], ndimloh, ndimgl, nex, ney, nnl, g, g2, nni, nnh, nnv;
} hercom;

/* Table of constant values */
static long int c__1 = 1;
static long int c__40 = 40;

//***************************************************************************
// Global variables
float k_nlin8;
```

```
double *z_ground_glob,h_force;

//***************************************************************************
// definition of the parameter structure
struct parameters
{
  float x0,y0,vx0,vy0; // initial conditions
  float t_max; // integration time
  float t_save; // interval between saves of data to disk
  int do_lyap;  // switch to toggle lyapunov calc on/off
  float eps_lyap; // separation between intial trajectories for lyapunov calc
  float k_nlin; // non-linearity parameter while testing with classical eqs.
  long max_pts; // max # of points computed in trajectory
  long pt_count; // counter for actually saved # of points
  float h_min; // minimum time step admissible in integrations
  float h_ini; // trial initial time step in integrations
  float eps_integ; // error tolerance factor for numerical integrator
  float h_force; // step size for computing the quantum force
  int pot_type; // classical(0) or quantum(1) potential
}; // end of struct. definition

//****************************************************************************
// initialize all parameters of simulation
void init_params(int argc,char **argv,parameters &p)
{
  int nargs=11; // # of args with argc convention: name+parameters=nargs

  // check for correct number of command-line arguments
  if(argc !=nargs){
    cerr << "\n*** ERROR ***\nImproper number of arguments.\n";
    cerr << "Calling sequence is:\n\n";
    cerr << " > qc_oscill.x  x0 y0 vx0 vy0 t_max t_save do_lyap "\
 << "pot_type k_nlin eps_integ\n";
    cerr << "   pot_type: 0 -> Classical, 1 -> Quantum\n\n";
    cerr << "(k_nlin is ignored for quantum pot, read from data file)\n\n";
    cerr << "Aborting...\n\n";
    exit(0);
  }
  // read command-line arguments
  p.x0=atof(argv[1]);
  p.y0=atof(argv[2]);
  p.vx0=atof(argv[3]);
  p.vy0=atof(argv[4]);
  p.t_max=atof(argv[5]);
  p.t_save=atof(argv[6]);
  p.do_lyap=atoi(argv[7]);
  p.pot_type=atoi(argv[8]);
  p.k_nlin=atof(argv[9]);
  p.eps_integ=atof(argv[10]); // error tolerance factor
  // these parameters depend on the previous values:
  p.h_ini=p.t_max/1000.0;  // trial initial time step
  // "hard-wired"  parameters
  p.eps_lyap=1e-5;
  p.h_min=1e-10;  // minimum admissible time step
  //p.eps_integ=1e-5;  // error tolerance factor
  p.max_pts=10000;
  p.h_force=0.1;
  // print parameters to screen
  fprintf(stderr,"\nCLASSICAL-QUANTUM OSCILLATOR\n");
  fprintf(stderr,"==========================\n\n");
  fprintf(stderr,"Input parameters:\n");
  fprintf(stderr,"  x0=%g, y0=%g, vx0=%g, vy0=%g,t_max=%g, t_save=%g\n",
  p.x0,p.y0,p.vx0,p.vy0,p.t_max,p.t_save);
  fprintf(stderr,"  do_lyap=%d, k_nlin(class)=%g, eps_integ=%g\n",
  p.do_lyap,p.k_nlin,p.eps_integ);
  fprintf(stderr,"Dependent parameters:\n");
  fprintf(stderr,"  h_ini=%g\n",p.h_ini);
  fprintf(stderr,"\"Hard wired\" parameters:\n");
  fprintf(stderr,"  eps_lyap=%g, h_min=%g, h_force=%g\n\n",
  p.eps_lyap,p.h_min,p.h_force);
} // end of init_params()

//*************************************************************************
/* ... Lee los parAmetros de cAlculo*/
void init_qpot_params(char nomarch_param[],parameters par)
{
  ifstream arch_param;
  double dif_k;
  char sp[]="        ";

  /* ... lectura de datos de disco */
  arch_param.open(nomarch_param);
  if (!arch_param) arch_error(nomarch_param);
  arch_param >> hercom.nnl >> hercom.g >> hercom.nex >> hercom.ney \
     >> param.ancho >> param.xmaxi >> param.dista \
     >> param.emass >> param.kq_nlin ;
  arch_param.close();
  /* ... Dimensiones de la region de calculo rectangular */
  param.xmaxi = param.ancho / 2.;
  param.dista = param.ancho / 2.;
  /* ... Se escribe en pantalla */
  cerr << "Quantum solution data:\n\n"
       << "#nodloc  #dof  #elem_x  #elem_y    width     "
       << "xmax     dist     mass    k_nlin\n\n";
  cerr << hercom.nnl << sp << hercom.g << sp << hercom.nex << sp
       << hercom.ney << sp << param.ancho << sp << param.xmaxi << sp
       << param.dista << sp << param.emass << sp << param.kq_nlin << "\n\n";
  /* ... Dimensiones */
  hercom.g2 = hercom.g * hercom.g;
  hercom.nnh = hercom.nex * (hercom.nnl - 1) + 1;
  hercom.nnv = hercom.ney * (hercom.nnl - 1) + 1;
  hercom.nni = hercom.nnh * hercom.nnv;
  hercom.ndimgl = hercom.g2 * hercom.nni;
  hercom.ndimloh = hercom.nnl * hercom.g;
  // Info on potentials
  dif_k=fabs(param.kq_nlin-par.k_nlin);
  if(dif_k>1e-6)
    fprintf(stderr,"*** WARNING: Classical k=%g diff. from quantum k=%g by %g\n",
    par.k_nlin,param.kq_nlin,dif_k);
  cerr << "Working with potential: ";
  if(par.pot_type == 0) cerr << "Classical. Fast!\n";
  else cerr << "Quantum. Slow...\n";
} /* init_qpot_params() */
```

```
//***************************************************************************
/* ... Lee los datos del estado base */
void input_ground_state(char nomarch_dat[],double *z)
{
  ifstream arch_dat;
  long int i1;
  static long int j;

  /* ... Se lee el vector propio en "vectbase.dat" */
  arch_dat.open(nomarch_dat);
  if(!arch_dat) arch_error(nomarch_dat);
  i1 = hercom.ndimgl;
  for (j = 0; j < i1; ++j) {
    arch_dat >> z[j];
  }
  arch_dat.close();
} /* input_ground_state() */

//***************************************************************************
/* ... Transformacion de las coordenadas del problema a las */
/* ... coordenadas escaleadas en cada elemento              */
void esp_ele(float *xe, float *ye, double *xee, double *yee,
     double *x, double *y, long int *tx, long int *ty, long int *t)
{
  static long int modx, mody;
  static double xori, yori;
  static long int numx, numy;
  static double ex, ey, xt, yt, xot, yot;

  xori = param.xmaxi;
  yori = param.dista;
  ex = param.xmaxi * 2 / hercom.nex;
  ey = param.ancho / hercom.ney;
  *xee = *xe + xori;
  *yee = -(*ye) + yori;
  numx = (long int) (*xee / ex);
  numy = (long int) (*yee / ey);
  modx = (long int) ((*xee / ex - numx) * 10);
  mody = (long int) ((*yee / ey - numy) * 10);
  *tx = numx + 1;
  *ty = numy + 1;
  if (modx == 0 && numx != 0) {
    *tx = numx;
  }
  if (mody == 0 && numy != 0) {
    *ty = numy;
  }
  *t = hercom.nex * (*ty - 1) + *tx;
  xot = (*tx - 1) * ex;
  yot = (*ty - 1) * ey;
  xt = *xee - xot;
  yt = *yee - yot;
  *x = xt * 2 / ex - 1;
  *y = yt * 2 / ey - 1;
} /* esp_ele() */

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

```
/* ... Polinomios de interpolacion de Hermite y sus primeras derivadas */
/* ... en punto (x,y), nodo local "il", grado de libertad "k"          */
void polinom(double *x, double *y, long int *k, long int *il,
     double *pihx, double *pihy, double *dpihx, double *dpihy)
{
  extern double dpih(long int *, double *);
  extern long int nconmatl(long int *, long int *);
  static long int i, ix, iy;
  extern double pih(long int *, double *);

  i = nconmatl(k, il);
  numfico(&i, &hercom.ndimloh, &ix, &iy);
  *pihx = pih(&ix, x);
  *pihy = pih(&iy, y);
  *dpihx = dpih(&ix, x);
  *dpihy = dpih(&iy, y);
} /* polinom() */

//***************************************************************************
/* ... Indice global del grado de libertad "k" (k=1,...g2) en         */
/* ... el nodo local "i" del elemento "t"                             */
/* ... nconmatN(i,t): numero global de nodo local "i" en elemento "t" */
/* ... g2=g*g: numero de grados de libertad en cada nodo              */
long int nconmatm(long int *k, long int *i, long int *t)
{
  long int ret_val;
  static long int ng, gx, gy, nx, ny;
  extern long int nconmatn(long int *, long int *);

  ng = nconmatn(i, t);
  numfico(&ng, &hercom.nnh, &nx, &ny);
  numfico(k, &hercom.g, &gx, &gy);
  ret_val = ((ny - 1) * hercom.g + gy - 1) *
    numline(&hercom.nex, &hercom.ndimloh, &hercom.g) +
    (nx - 1) * hercom.g + gx;
  return ret_val;
} /* nconmatm() */

//***************************************************************************
/* ... Numero global del nodo local "i" del elemento "t" */
/* ... (ix,iy): (columna,fila) del nodo local "i"        */
/* ... (tx,ty): (columna,fila) del elemento "t"          */
/* ... nnh: numero de nodos en cada fila global          */
/* ... nnl: numero de nodos horizontales y verticales en cada elemento */
long int nconmatn(long int *i, long int *t)
{
  /* System generated locals */
  long int ret_val, i1, i2, i3, i4, i5, i6, i7, i8;
  /* Local variables */
  extern long int mcero(long int *);
  static long int ix, iy, tx, ty;

  numfico(i, &hercom.nnl, &ix, &iy);
  numfico(t, &hercom.nex, &tx, &ty);
  i1 = ty - 1;
  i2 = ty - 1;
  i3 = iy - 1;
  i4 = ty - 1;
  i5 = ty - 1;
  i6 = iy - 2;
  i7 = tx - 1;
  i8 = tx - 1;
  ret_val = (numline(&i1, &hercom.nnl, &c__1) - mcero(&i2) *
     (1-mcero(&i3)) + (1 - mcero(&i4)) * (iy - 1) + mcero(&i5)*
     mcero(&i6) * (iy - 2)) * hercom.nnh +
    numline(&i7, &hercom.nnl, &c__1) + ix - mcero(&i8);
  return ret_val;
} /* nconmatn() */

//*************************************************************************
/* ... Numero del grado de libertad "k" en el nodo local "i" */
/* ... en la malla dentro de un elemento                     */
/* ... (ix,iy): (columna,fila) del nodo local "i"            */
/* ... (tx,ty): (columna,fila) del elemento "t"              */
/* ... nnh: numero de nodos en cada fila global              */
/* ... nnl: numero de nodos horizontales y verticales en cada elemento */
long int nconmatl(long int *k, long int *i)
{
  long int ret_val;
  static long int ix, iy, kx, ky;

  numfico(i, &hercom.nnl, &ix, &iy);
  numfico(k, &hercom.g, &kx, &ky);
  ret_val = ((iy - 1) * hercom.g + ky - 1) * hercom.ndimloh + (ix - 1) *
    hercom.g + kx;
  return ret_val;
} /* nconmatl() */

//*************************************************************************
/* Potencial cuantico en el punto (xe,ye).
   Esta rutina calcula no solo qpot, sino tambien psi, dxpsi, dypsi. */
void qpot(float *xe, float *ye, double *z,  double *potc,
       double *psi, double *dxpsi, double *dypsi)
{
  long int i1, i2;
  double d1, d2, d3, d4, d5;
  static long int k, t, il, im, tx, ty;
  static double pihx, pihy, dpihx, dpihy;
  static double x, y, hx, hy, xee, yee, z_im;
  extern long int nconmatm(long int *, long int *, long int *);
  extern void esp_ele(float *, float *, double *, double *, double *, double *,
      long int *,long int *, long int *);
  extern void  polinom(double *, double *, long int *, long int *, double *,
       double *, double *, double *);
  /* Parameter adjustments */
  --z;
  /* Function Body */
  hx = param.xmaxi * 2 / hercom.nex / (hercom.nnl - 1);
  hy = param.ancho / hercom.ney / (hercom.nnl - 1);
  esp_ele(xe, ye, &xee, &yee, &x, &y, &tx, &ty, &t);
  *psi = 0.;
  *dxpsi = 0.;
  *dypsi = 0.;
  i1 = hercom.nnl * hercom.nnl;
  for (il = 1; il <= i1; ++il) {
    i2 = hercom.g2;
    for (k = 1; k <= i2; ++k) {
      im = nconmatm(&k, &il, &t);
      polinom(&x, &y, &k, &il, &pihx, &pihy, &dpihx, &dpihy);
      z_im = z[im];
      *psi += z_im * pihx * pihy;
      *dxpsi += z_im * dpihx * pihy;
      *dypsi += z_im * pihx * dpihy;
    }
  }
  /* Computing 2nd powers */
  d1 = *dxpsi;
  d2 = hx;
  d3 = *dypsi;
  d4 = hy;
  d5 = *psi;
  *potc = (d1 * d1 / (d2 * d2) + d3 * d3 / (d4 * d4)) /
    (param.emass * 2 * (d5 * d5));
} /* qpot() */

//****************************************************************************
// compute the "quantum force" by numerically finding the gradient of the potential
void qforce(float x, float y, double *z,
    double *fx, double *fy,double h)
{
  float h2,xp,xm,yp,ym;
  double vpx,vmx,vpy,vmy;
  double psi, dxpsi, dypsi;

  // force at the origin is always zero
  if(x*x+y*y < 1e-5 ){
    *fx = *fy = 0;
    return;
  }
  // use a centered two-point formula for the derivatives
  h2=2.0*h;
  xp = x+h; xm = x-h;
  yp = y+h; ym = y-h;
  qpot(&xp,&y,z,&vpx,&psi,&dxpsi,&dypsi);
  qpot(&xm,&y,z,&vmx,&psi,&dxpsi,&dypsi);
  qpot(&x,&yp,z,&vpy,&psi,&dxpsi,&dypsi);
  qpot(&x,&ym,z,&vmy,&psi,&dxpsi,&dypsi);
  *fx=(vmx-vpx)/h2;
  *fy=(vmy-vpy)/h2;
} // qforce()

//****************************************************************************
/* ... .........................................................     */
/* ... A continuacion vienen rutinas "de utilidad": inicializacion     */
/* ... de matrices, ordenamiento, funciones numericas varias,          */
/* ... integracion gaussiana y polinomios de interpolacion de Hermite. */
/* ... .........................................................     */

//****************************************************************************
/* ... Polinomios de interpolacion de Hermite               */
/* ... pih(i,x)=sum(b(k,i)*x^(k-1),k=1..ndimloh)            */
/* ... nnl: numero de nodos locales en [-1,1], 1 < nnl <= 10 */
/* ... g: numero de grados de libertad, 1 <= g <= 4          */
/* ... ndimloh=g*nnl: numero de polinomios                   */
/* ... ndimloh-1: grado de los polinomios                    */
/* ... Polinomio de interpolacion de Hermite                 */
double pih(long int *ii, double *x)
{
#define b_ref(a_1,a_2) hercom.b[(a_2)*40 + a_1 - 41]
  long int i1;
  static long int k;
  static double y, pol;

  pol = 0.;
  y = 1.;
  i1 = hercom.ndimloh;
  for (k = 1; k <= i1; ++k) {
    pol += b_ref(k, *ii) * y;
    y *= *x;
  }
  return pol;
} /* pih() */
#undef b_ref

//****************************************************************************
/* ... Derivada de polinomios de interpolacion de Hermite    */
/* ... dpih(i,x)=sum((k-1)*b(k,i)*x^(k-2),k=2..ndimloh)      */
/* ... nnl: numero de nodos locales en [-1,1], 1 < nnl <= 10 */
/* ... g: numero de grados de libertad, 1 <= g <= 4          */
/* ... ndimloh=g*nnl: numero de polinomios                   */
/* ... ndimloh-1: grado de los polinomios                    */
/* ... Derivada de polinomio de interpolacion de Hermite     */
double dpih(long int *ii, double *x)
{
#define b_ref(a_1,a_2) hercom.b[(a_2)*40 + a_1 - 41]
  long int i1;
  static long int k;
  static double y, pol;

  pol = 0.;
  y = 1.;
  i1 = hercom.ndimloh;
  for (k = 2; k <= i1; ++k) {
    pol += (k - 1) * (b_ref(k, *ii) * y);
    y *= *x;
  }
  return pol;
} /* dpih() */
#undef b_ref

//****************************************************************************
/* ... Se evaluan los coeficientes "b(k,i)" que definen los  */
/* ... polinomios de interpolacion de Hermite                */
/* ... pih(i,y)=sum(b(k,i)*y^(k-1),k=1..ndimloh)             */
/* ... nnl: numero de nodos locales en [-1,1], 1 < nnl <= 10 */
/* ... g: numero de grados de libertad, 1 <= g <= 4          */
/* ... ndimloh=g*nnl: numero de polinomios                   */
/* ... ndimloh-1: grado de los polinomios                    */
/* ... Vector de nodos locales                              */
void polinthe()
{
  extern void nodpih(), matpih(), invmat();

  nodpih();
  matpih();
  invmat();
} /* polinthe() */

//**************************************************************************
/* ... Polinomios de interpolacion de Hermite          */
/* ... Vector {zn} de nodos locales de dimension "nnl" */
/* ... vector de nodos locales                         */
void nodpih()
{
  long int i1;
  static double h;
  static long int i;

  h = 2. / (hercom.nnl - 1);
  hercom.zn[0] = -.99999999999999989;
  i1 = hercom.nnl;
  for (i = 2; i <= i1; ++i) {
    hercom.zn[i - 1] = hercom.zn[i - 2] + h;
  }
} /* nodpih() */

//**************************************************************************
/* ... Polinomios de interpolacion de Hermite             */
/* ... Matriz [a] de dimension ndimloh*ndimloh            */
/* ... formada por repeticion de "g" filas de la forma    */
/* ... [1, y, y^2,   y^3,...                   y^(g*n-1)] */
/* ... [0, 1, 2*y, 3*y^2,...           (g*n-1)*y^(g*n-2)] */
/* ... [0, 0,   2,   6*y,...   (g*n-1)*(g*n-2)*y^(g*n-3)] */
/* ... [.  .    .      . ...                           .] */
/* ... [.  .    .      . ...                           .] */
/* ... [.  .    .      . ...                           .] */
/* ... [0, 0,   0,     0,... (g*n-1)!/(g*n-g)!*y^(g*n-g)] */
/* ... "y" evaluada en zn(i), para i=1..n                 */
/* ... Matriz de y^k y sus derivadas en los nodos         */
void matpih()
{
#define a_ref(a_1,a_2) hercom.a[(a_2)*40 + a_1 - 41]
  long int i1, i2, i3, i4, i5;
  static long int i, j, k, l;

  i1 = hercom.ndimloh;
  i2 = hercom.g;
  for (i = hercom.g; i2 < 0 ? i >= i1 : i <= i1; i += i2)
    {
      l = i / hercom.g;
      i3 = hercom.g - 1;
      for (k = 0; k <= i3; ++k) {
i4 = hercom.ndimloh - 1;
for (j = 0; j <= i4; ++j) {
  i5 = j - k;
  a_ref((l - 1) * hercom.g + k + 1, j + 1) = coe(&j, &k) *
    pow(hercom.zn[l - 1], i5);
}
      }
    }
} /* matpih() */
#undef a_ref

//****************************************************************************
/* ... Inversa [b] de una matriz [a] de dimension ndimloh*ndimloh */
/* ... usa las rutinas "ludcmp" y "lubksb" de Numerical Recipes   */
/* ... La matriz de coeficientes [b] es la inversa de [a]         */
/* ... La columna "ii" de [b] contiene los coeficientes           */
/* ... de [1, y, y^2, y^3,...,y^(g*n-1)] en el polinomio de       */
/* ... interpolacion de Hermite, "pih(ii,y)"                      */
/* ... Matriz identidad                                           */
void invmat()
{
#define b_ref(a_1,a_2) hercom.b[(a_2)*40 + a_1 - 41]
  long int i1, i2;
  static double d;
  static long int i, j;

  i1 = hercom.ndimloh;
  for (i = 1; i <= i1; ++i) {
    i2 = hercom.ndimloh;
    for (j = 1; j <= i2; ++j) {
      b_ref(i, j) = 0.;
    }
    b_ref(i, i) = 1.;
  }
  ludcmp(hercom.a, &hercom.ndimloh, &c__40, hercom.indx, &d);
  i1 = hercom.ndimloh;
  for (j = 1; j <= i1; ++j) {
    lubksb(hercom.a,&hercom.ndimloh,&c__40,hercom.indx, &b_ref(1,j));
  }
} /* invmat() */
#undef b_ref

//****************************************************************************
// Runge-Kutta integrator from Numerical Recipes in C, 2nd ed, p.721
// Some changes have been made to original routine.
#define MAXSTP 5000000
#define TINY 1.0e-30

void odeint(float ystart[], int nvar, float x1, float x2, float eps, float h1,
    float hmin, long *nok, long *nbad,
    float *xp,float **yp,float dxsav,long kmax,long &kount,
    void (*derivs)(float, float [], float []),
    void (*rkqs)(float [], float [], int, float *, float, float,
 float [], float *, float *,
 void (*)(float, float [], float [])))
{
  long nstp,i;
  float xsav,x,hnext,hdid,h;
  float *yscal,*y,*dydx;
  yscal=vector(1,nvar);
  y=vector(1,nvar);
  dydx=vector(1,nvar);
  x=x1;
  h=SIGN(h1,x2-x1);
  *nok = (*nbad) = kount = 0;
  for (i=1;i<=nvar;i++) y[i]=ystart[i];
  if (kmax > 0) xsav=x-dxsav*2.0;
  for (nstp=1;nstp<=MAXSTP;nstp++) {
    (*derivs)(x,y,dydx);
    for (i=1;i<=nvar;i++)
      yscal[i]=fabs(y[i])+fabs(dydx[i]*h)+TINY;
    if (kmax > 0 && kount < kmax-1 && fabs(x-xsav) > fabs(dxsav)) {
      xp[++kount]=x;
      for (i=1;i<=nvar;i++) yp[i][kount]=y[i];
      xsav=x;
    }
    if ((x+h-x2)*(x+h-x1) > 0.0) h=x2-x;
    (*rkqs)(y,dydx,nvar,&x,h,eps,yscal,&hdid,&hnext,derivs);
    if (hdid == h) ++(*nok); else ++(*nbad);
    if ((x-x2)*(x2-x1) >= 0.0) {
      for (i=1;i<=nvar;i++) ystart[i]=y[i];
      if (kmax) {
xp[++kount]=x;
for (i=1;i<=nvar;i++) yp[i][kount]=y[i];
      }
      free_vector(dydx,1,nvar);
      free_vector(y,1,nvar);
      free_vector(yscal,1,nvar);
      return;
    }
    if (fabs(hnext) <= hmin) nrerror("Step size too small in odeint");
    h=hnext;
  }
  nrerror("Too many steps in routine odeint");
} // end of odeint()
#undef MAXSTP
#undef TINY

//******************************************************************************
// compute right hand side of differential equations for the purely classical
// potential
void rhs_class(float t,float *Q,float *dQdt)
{
  float alpha;
  alpha=k_nlin8*Q[1]*Q[2];
  // classical eqs of motion for V(x,y)=(x^2+y^2)/2+4kx^2y^2
  dQdt[1]= Q[3];
  dQdt[2]= Q[4];
  dQdt[3]= -Q[1]-alpha*Q[2];
  dQdt[4]= -Q[2]-alpha*Q[1];
} // end of rhs_class()

//******************************************************************************
// compute right hand side of differential equations for Lyapunov exponent case
void rhs_class_lyap(float t,float *Q,float *dQdt)
{
  float alpha1,alpha2;
  // classical eqs of motion for V(x,y)=(x^2+y^2)/2+4kx^2y^2
  alpha1=k_nlin8*Q[1]*Q[2];
  dQdt[1]= Q[3];
  dQdt[2]= Q[4];
  dQdt[3]= -Q[1]-alpha1*Q[2];
  dQdt[4]= -Q[2]-alpha1*Q[1];
  // similar dynamics for the second trajectory
  alpha2=k_nlin8*Q[5]*Q[6];
  dQdt[5]= Q[7];
  dQdt[6]= Q[8];
  dQdt[7]= -Q[5]-alpha2*Q[6];
  dQdt[8]= -Q[6]-alpha2*Q[5];
} // end of rhs_class_lyap()

//****************************************************************************
// compute right hand side of differential equations for the quantum case
void rhs_quant(float t,float *Q,float *dQdt)
{
  double fx,fy;

  qforce(Q[1],Q[2],z_ground_glob,&fx,&fy,h_force);
  dQdt[1]= Q[3];
  dQdt[2]= Q[4];
  dQdt[3]= float(fx);
  dQdt[4]= float(fy);
  // for debugging
  //printf("%8f\t%8f\t%8f\t%8f\n",Q[1],Q[2],fx,fy);
} // end of rhs_quant()

//****************************************************************************
// compute right hand side of differential equations for the quantum case,
// including Lyapunov exponent calculation
void rhs_quant_lyap(float t,float *Q,float *dQdt)
{
  double fx1,fy1,fx2,fy2;

  qforce(Q[1],Q[2],z_ground_glob,&fx1,&fy1,h_force);
  dQdt[1]= Q[3];
  dQdt[2]= Q[4];
  dQdt[3]= float(fx1);
  dQdt[4]= float(fy1);
  // similar dynamics for the second trajectory
  qforce(Q[5],Q[6],z_ground_glob,&fx2,&fy2,h_force);
  dQdt[5]= Q[7];
  dQdt[6]= Q[8];
  dQdt[7]= float(fx2);
  dQdt[8]= float(fy2);
} // end of rhs_quant_lyap()

//****************************************************************************
// integrate the equations of motion for one trajectory (no lyapunov exponent)
void integrate_one(parameters &par)
{
 long i,num_vars,nok,nbad;
 float t_ini,*q0,*t,**Q;
 // make data arrays
 num_vars=4;
 q0=vector(1,num_vars);
 t=vector(1,par.max_pts);
 Q=matrix(1,num_vars,1,par.max_pts);
 // initial conditions
 t_ini=0;
 q0[1]=par.x0;  q0[2]=par.y0;
 q0[3]=par.vx0; q0[4]=par.vy0;
 // set global vble needed by rhs_class
 k_nlin8=8.0*par.k_nlin;
 // integrate the eqs of motion
 // with classical or quantum potential
 if(par.pot_type==0) {
   printf("# Classical potential with k_nlin = %g\n",par.k_nlin);
   odeint(q0,num_vars,t_ini,par.t_max,par.eps_integ,par.h_ini,par.h_min,&nok,&nbad,
  t,Q,par.t_save,par.max_pts,par.pt_count,rhs_class,rkqs);
 }
 else {
   printf("# Quantum potential with k_nlin = %g\n",param.kq_nlin);
   odeint(q0,num_vars,t_ini,par.t_max,par.eps_integ,par.h_ini,par.h_min,&nok,&nbad,
  t,Q,par.t_save,par.max_pts,par.pt_count,rhs_quant,rkqs);
 }
 // show output data
 printf("# x(t)\t\ty(t)\n");
 for(i=1;i<=par.pt_count;i++) {
   printf("%8g\t%8g\n",Q[1][i],Q[2][i]);
 }
 // free data arrays
 free_vector(q0,1,num_vars); free_vector(t,1,par.max_pts);
 free_matrix(Q,1,num_vars,1,par.max_pts);
} // end of integrate_one()

//****************************************************************************
// integrate the equations of motion for two trajectories, get Lyapunov exp.
// both orbits are integrated simultaneously, with slightly different initial conds.
void integrate_lyap(parameters &par)
{
 long i,num_vars,nok,nbad;
 float t_ini,*q0,*t,**Q,*delta;

 // make data arrays
 num_vars=8; // 8 because we're integrating TWO trajectories
 q0=vector(1,num_vars);
 t=vector(1,par.max_pts);
 delta=vector(1,par.max_pts);
 Q=matrix(1,num_vars,1,par.max_pts);
 // initial conditions
 t_ini=0;
 q0[1]=par.x0;
 q0[5]=q0[1]+par.eps_lyap;
 q0[6]=q0[2]=par.y0;
 q0[7]=q0[3]=par.vx0;
 q0[8]=q0[4]=par.vy0;
 // set global vble needed by rhs_class
 k_nlin8=8.0*par.k_nlin;
 // integrate the eqs of motion
 // with classical or quantum potential
 if(par.pot_type==0) {
   printf("# Classical potential with k_nlin = %g\n",par.k_nlin);
   odeint(q0,num_vars,t_ini,par.t_max,par.eps_integ,par.h_ini,par.h_min,&nok,&nbad,
  t,Q,par.t_save,par.max_pts,par.pt_count,rhs_class_lyap,rkqs);
 }
 else {
   printf("# Quantum potential with k_nlin = %g\n",param.kq_nlin);
   odeint(q0,num_vars,t_ini,par.t_max,par.eps_integ,par.h_ini,par.h_min,&nok,&nbad,
  t,Q,par.t_save,par.max_pts,par.pt_count,rhs_quant_lyap,rkqs);
 }
 // compute *phase space* separation of orbits and show output data
 // print both orbits and delta(t): < x1(t), y1(t), x2(t), y2(t), t, delta(t) >
 printf("# x1(t)\t\ty1(t)\t\tx2(t)\t\ty2(t)\t\tt\t\tdelta(t)\n");
 for(i=1;i<=par.pt_count;i++) {
   delta[i]=sqrt(SQR(Q[5][i]-Q[1][i])+SQR(Q[6][i]-Q[2][i])+
 SQR(Q[7][i]-Q[3][i])+SQR(Q[8][i]-Q[4][i]));
      printf("%8g\t%8g\t%8g\t%8g\t%8g\t%8g\n",
  Q[1][i],Q[2][i],Q[5][i],Q[6][i],t[i],delta[i]);
 }
 // free data arrays
 free_vector(q0,1,num_vars); free_vector(t,1,par.max_pts);
 free_vector(delta,1,par.max_pts);
 free_matrix(Q,1,num_vars,1,par.max_pts);
} // end of integrate_lyap()

//******************************* main program *******************************
int main(int argc, char *argv[])
{
  double *z_ground;
  parameters par;

  // read all initial parameters and data
  init_params(argc,argv,par);
  init_qpot_params("anhosc.ini",par);
  z_ground=(double *)malloc((size_t) (hercom.ndimgl*sizeof(double)));
  input_ground_state("vectbase.dat",z_ground);
  // set global vbles
  z_ground_glob=z_ground;
  h_force=par.h_force;
  /* ... Evaluacion de los coeficientes "b(k,i)" */
  polinthe();
  if(par.do_lyap)integrate_lyap(par);
  else integrate_one(par);
  free(z_ground);
  cerr << "DONE.\n\n";
  return 0;
} /* end of main() */
//******************** END OF FILE <qc_oscill.cc> **************************
#undef NRANSI
```

References

[1] G. Casati, Chaos **6**, 391 (1996)

[2] M.C. Gutzwiller, *Chaos in Classical and Quantum Mechanics* (Springer-Verlag, Berlin, 1990)

[3] D. Bohm, Phys. Rev. **85,** 166 (1952), Phys. Rev. **85,** 180 (1952), Phys. Rev. **89,** 458 (1953)

[4] R. Vilela, Phys. Lett. A **187**, 299 (1994)

[5] F.H.M. Faisal and U. Schwengelbeck, Phys. Lett. A **199**, 281 (1995)

[6] F.H.M. Faisal and U. Schwengelbeck, Phys. Lett. A **207**, 31 (1995)

[7] Z. Malik, preprint quant-ph/9506026 at http://xxx.lanl.gov

[8] K. Berndl et. al., Nuovo. Cim. **110B**, 737 (1995)

[9] G. Iacomelli and M. Pettini, Phys. Lett. A **212**, 29 (1996)

[10] G. Garcia, Phys. Rev. A **53**, 2059 (1996)

[11] G. Junker, *Supersymmetic Methods in Quantum and Statistical Physics* (Springer-Verlag, Berlin, 1996)

[12] A. Comtet, A.D. Bandrauk and D. Campbell, Phys. Lett. **150B**, 159 (1985)

[13] L.R. Ram-Mohan, S. Saigal and J. Shertzer, Comp. Phys. Jan. 1990 50 (1990)

[14] W.H. Press et. al., *Numerical Recipes in C,* 2^{nd} edition (Cambridge University Press, Cambridge, 1992)

[15] H-D. Meyer, J. Chem. Phys. **84**, 3147 (1986)

[16] M. Hénon, Physica **D5**, 412 (1982)

[17] V.I. Arnol'd, *Mathematical Methods of Classical Mechanics* Appendix 10 (Springer-Verlag, New York, 1978)