Rank-1 Saddle Transport in Three or More Degrees of Freedom Scattering Reactions

Steven Brunton,
Senior Thesis Advisors[†]*: Jerrold Marsden & Wang Sang Koon*

Control and Dynamical Systems
California Institute of Technology, Pasadena, CA 91125

A computationally effective method is developed to study rank-1 saddle transport. In particular, the lifetime distribution is calculated for the planar, nonzero angular momentum scattering reaction of H_2 with H_2O. The scattering model uses the approximate potential of Wiesenfeld *et al.*[14], which has 3 essential degrees of freedom (DOF) and a rotating transition state (TS) at physically relevant energies. A Monte Carlo sampling of the energy surface on a transverse cut near the TS provides a representative set of trajectories which are integrated forward and backward to test for reactivity. Reactive trajectories are then integrated forward into the bound state until they escape the reaction, resulting in orbit/collision data for each trajectory. This data is collected and binned until the standard deviation of the data in each bin is within a user defined tolerance, yielding a lifetime distribution with bounded error terms. To test this method, it is applied to the well studied phenomena of electron scattering in the Rydberg atom with crossed fields[1,4] and compared with the literature.

. . . . . .

Contents

[†]Since I have had the privilege to join the Marsden group over three years ago, I have been blessed with the most fulfilling learning experiences of my time at Caltech. I am grateful for my advisors Jerrold Marsden and Wang Sang Koon, whose insights and guidance have sharpened my mathematical intuition and set me on my current research path. Additional thanks are due to Frederic Gabern, Shane Ross, Katalin Grubits, Tomohiro Yanao, Philip Du Toit, Nawaf Bou-Rabee, Bingni Wen, and James Martin for valuable discussions, ideas and generous help along the way.

List of Figures

1 Introduction

A scattering reaction between two molecules is a chemical reaction in the sense that the molecules come close enough to interact, transfer momentum, and leave in a different state than they entered. A scattering reaction has two potential energy wells corresponding to *bound* and *unbound* states which are separated in phase space by a rank-1 saddle[†]. The transition state (TS) opens at energies slightly larger than that of the saddle and connects the bound and unbound states. In this way, the TS is an energetic bottleneck through which reactants pass to become products.

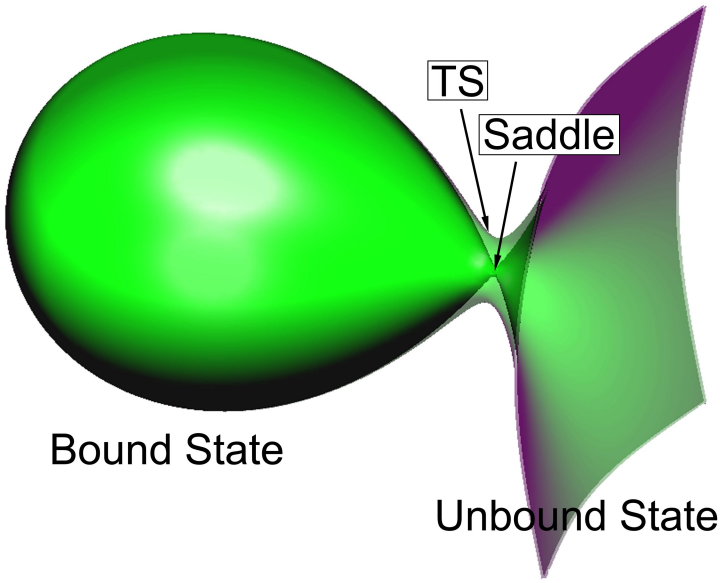

Figure 1: Schematic energy surface for a scattering reaction. A surface which separates permissible configurations from those which are energetically forbidden is known as a **Hill's region**.

Scattering reactions are typically studied with total angular momentum equal to zero, $J = 0$. Experimentally, however, the importance of nonzero J is known in the context of collision-induced absorption. Polar molecules, such as water, absorb light differently with different rotational velocities. For example, when water and hydrogen molecules collide and transfer angular momentum, the absorption spectrum of water changes. Absorption shifts caused by the collision of water and hydrogen molecules can therefore provide valuable clues about the density and energy distribution of interstellar gas clouds[12,14].

There are no fixed points in a laboratory frame for a planar scattering reaction with nonzero angular momentum; therefore, the scattering TS is located near a rank-1 saddle of a *reduced* system. The reduced system is obtained by using conservation of angular momentum to work on a γ-angular momentum level set. The equations of motion are then written in body frame coordinates[‡] which neglect internal motions of the molecules and have 3 degrees of freedom (DOF), an

[†]rank-1 saddle - fixed point with spectrum saddle$\times$center$\times\cdots\times$center.

[‡]body frame coordinates - centered on and oriented about one of the molecules.

improvement over the unreduced 4DOF system. The rank-1 saddle underlying the scattering TS is thus a fixed point of the reduced system.

1a Non-RRKM Lifetime Distributions

Transition state theory (TST), a workhorse for the chemical community, typically operates on the simplifying assumption of an unstructured phase space[11]. Under this assumption, the reaction rate for the length of a scattering reaction is reflected by the ratio of flux across the TS to the phase space volume of the bound state. This estimation leads to a RRKM (Rice-Ramsperger-Kassel-Marcus) lifetime distribution for the expectation of bound times, characterized by exponential decay. It is well known, however, that even chaotic phase space is structured. Reaction dynamics through a TS near a rank-1 saddle are mediated by invariant manifold tubes associated with periodic orbits around the saddle[8,13]. These Conley-McGehee tubes[†] have been used to guide the Genesis Discovery craft into halo orbit around the L_2 saddle point of the Sun-Earth Restricted 3-Body Problem (R3BP)[7]. Tube dynamics have also been merged with invariant set methods and Monte Carlo methods to exhibit non-RRKM lifetime distributions in the isomerization of the Rydberg atom with crossed fields[1,4].

Previous efforts to study transport in a structured phase space have involved high order normal forms or invariant set methods to directly compute transport rates using the volumes of transport tube intersections. Points in the intersection of incoming and outgoing tubes correspond to reactive trajectories which become bound via the entrance tube, and escape via the exit tube. In the case of scattering reactions, molecular species are preserved during and after the reaction; the reaction occurs when molecules enter a bound state and then escape back to the unbound state after some interaction. The relative volumes of the intersections of the entrance tube after n revolutions with the exit tube in the bound state provides a structured estimate of the lifetime distribution for the number of revolutions in the bound state.

Instead of computing transport tube intersections directly, this method uses a Monte Carlo random sampling to find a representative set of reactive trajectories which are evenly distributed on the energy surface. Each reactive trajectory is flowed forward in time until its escape, after which the bound time and either the number of nuclear core orbits or core collisions, depending on the nature of the scattering, are recorded and binned. Because these trajectories are selected at random, the lifetime distribution obtained from the orbit/collision data not only incorporates the structure of phase space, but converges with a small number of trajectories as well.

1b Rank-1 Saddle Transport

A rank-1 saddle is a fixed point with eigenvalues: $\pm\lambda, \pm\sqrt{-1}\omega_1, \cdots, \pm\sqrt{-1}\omega_{n-1}$, meaning it is a saddle in one direction and a center in every other direction. In linearized coordinates around a rank-1 saddle of an n-DOF Hamiltonian vector field, the Hamiltonian may be written in quadratic

[†] invariant manifold tubes associated with rank-1 saddles are known as Conley-McGehee tubes.

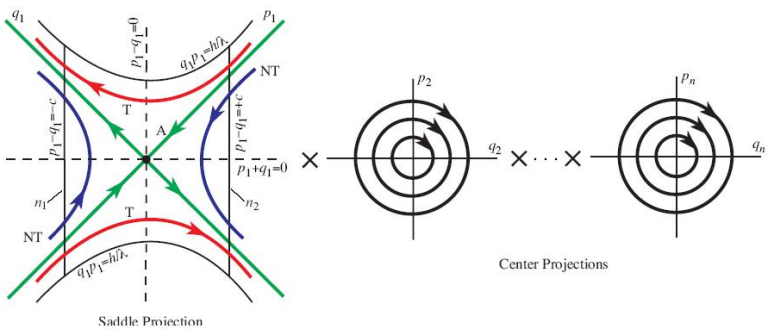

Figure 2: Projection of rank-1 saddle dynamics onto saddle and center directions.

normal form as:

$$H_2(q_1, q_2, \ldots, q_n, p_1, p_2, \ldots, p_n) = \lambda q_1 p_1 + \frac{\omega_1}{2}\left(q_2^2 + p_2^2\right) + \ldots + \frac{\omega_{n-1}}{2}\left(q_n^2 + p_n^2\right)$$

where (q_1, p_1) may be thought of as the reaction coordinate (saddle direction) and $(q_2, p_2), \ldots, (q_n, p_n)$ are bath modes (center directions). For energy h above the saddle, fixing $0 \neq \lambda q_1 p_1 < h$ leaves $h - \lambda q_1 p_1 = C$ energy for distribution among the bath modes. The equation $C = \sum_{i=1}^{n-1} \frac{\omega_i}{2}\left(q_{i+1}^2 + p_{i+1}^2\right)$ defines a surface that is homeomorphic to S^{2n-3}, the $(2n-3)$-sphere. Over the continuum of $\lambda q_1 p_1$ between 0 and h, these spheres plus the point at $\lambda q_1 p_1 = h$ add to form a surface homeomorphic to $D^{2(n-1)}$, the filled $(2n-3)$-sphere. It has been shown that the surfaces $D^{2(n-1)}$ associated with $q_1, p_1 \geq 0$ and $q_1, p_1 \leq 0$ are the *footprints* of the entrance and exit tubes, respectively[8].

2 Methods

The method presented here is adapted from the method of Gabern *et al.* which merges tube dynamics and Monte Carlo sampling methods to compute non-RRKM lifetime distributions for electron scattering of the Rydberg atom. This new method addresses a number of issues such as the numerical difficulty of computing with high order normal forms and the need for error estimates on lifetime distribution data.

There are three stages to compute the lifetime distribution for a scattering reaction: 1.) find the TS near a saddle & the Hill's region (bound and unbound states), 2.) identify a representative selection of reactive trajectories, and 3.) send them into the bound region and collect data until they escape the reaction. The previous method utilizes a high order normal form expansion in stage 2 to determine the D^4 *footprint* of the entrance tube ($D^4 \times I$), from which reactive trajectories are randomly sampled. Reactive trajectories are represented by phase space points which integrate forward along the vector field into the bound region, and backward along the vector field into the unbound region. This method has been proven to work well for low (≤ 3) DOF systems, but the normal form computation becomes exponentially difficult with higher DOF so that identifying reactive trajectories soon becomes the limiting computation.

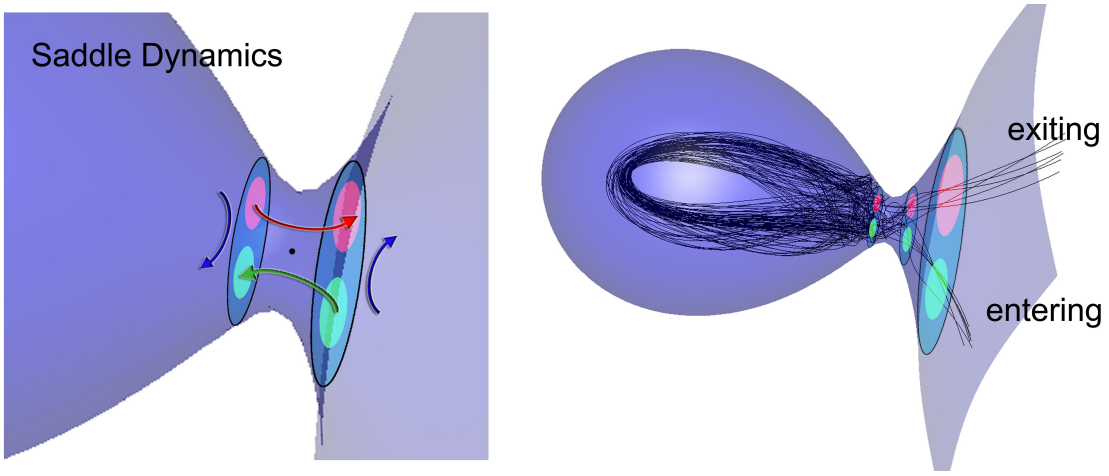

Figure 3: Nonlinear saddle dynamics near the TS for Rydberg scattering reaction.

As mentioned above, the invariant manifold entrance tube for an n-DOF model is homeomorphic to $D^{2(n-1)} \times I$, with I parametrizing the reaction coordinate (locally, the saddle direction). Therefore, the intersection of the tube with a nearby transverse cut[†] is topologically $D^{2(n-1)}$, a compact surface. Because $D^{2(n-1)}$ is compact, it may be bounded by a box in phase space.

This method requires a phase space box which tightly bounds the set of all reactive trajectories on a transverse cut. Rather than using the normal form computation to find the $D^{2(n-1)}$ *footprint*, points are selected at random from inside the bounding box and projected onto the energy

[†]transverse to the reaction coordinate I

surface. The behavior of these trajectories quickly forms a statistically representative distribution for the behavior of all such trajectories. If the transverse cut is made in the unbound state, trajectories are tested for reactivity by integration forward until they either intersect a transverse cut in the bound state (reactive) or a cut farther into the unbound state (unreactive).

2a Bounding Box Methods at Transverse Cut

To avoid trajectories which integrate both forward *and* backward into the bound region (corresponding to nontransit trajectories), the cut C_{out} is to be chosen sufficiently far from the saddle. The algorithm used to obtain a bounding box at C_{out} proceeds as follows: 1.) find a rough box around reactive trajectories at C_{fp}, 2.) select a handful of trajectories from rough box that integrate forward into C_{in} and backward into C_{out}, ensuring reactivity of the trajectory. The intersection of these trajectories with C_{out} gives a rough box at C_{out}. Finally, 3.) refine the rough boxes at each step, growing them by test sampling in the border region.

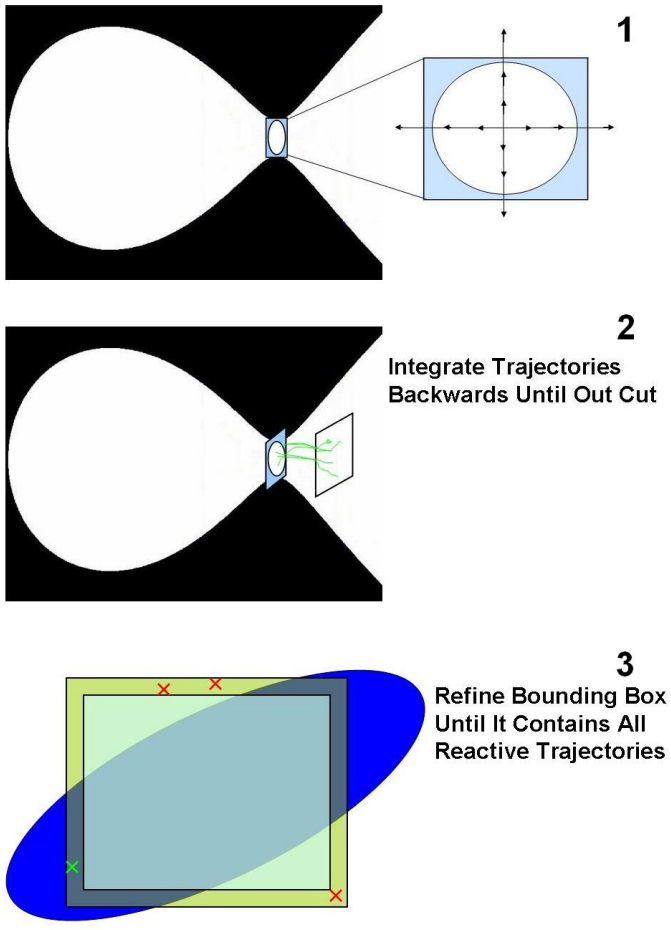

Figure 4: Three stages involved in determining the bounding box at C_{out}.

The Monte Carlo sampling method provides an unbiased sample from the bounding box. This sample of points is then projected, via momentum variables, onto the energy surface; however, because the bounding box is larger than the energy surface, some sample points project away from the energy surface in every momentum direction. Therefore, a tighter bounding box results in more efficient sampling. For example, if a tightly fitting box is expanded by 5% in each of its five dimensions, then the volume of the new box is $1.05^5 \approx 1.28$ times larger than necessary. This means that at least $\frac{.28}{1.28} \approx 22\%$ of points randomly selected in the box will not project onto the energy surface. If the box is twice too large in each dimension then at least $\frac{31}{32} \approx 97\%$ of randomly selected points will not project onto the energy surface. The need for a tight-fitting bounding box becomes more pronounced with higher dimensional energy surfaces.

2b Monte Carlo Sampling for Reactive Trajectories

After obtaining a tight-fitting bounding box at C_{out}, individual reactive trajectories may be found and tested as follows: select points at random from inside the box until one projects onto the relevant energy surface, and continue until one of these trajectories integrates forward into the cut C_{in}, identifying it as reactive. Once a reactive trajectory is obtained, run it through the bound region until it escapes, recording its specific behavior (e.g., $\#$ of orbits/collisions in bound state).

Because the reactive trajectories are not chosen from a grid, but are randomly distributed on the energy surface, each trajectory's contribution to the lifetime distribution is independent from that of any other trajectory. By binning the data collected, and running a *bundle* of trajectories (say 100 or 1000 at a time) through the bound state, the average value of a bin for each bundle of data may be compared with the average value for the entire data set. The standard deviation for each bin value among the various bundles may be calculated according to the formula:

$$\mathrm{SD}(i) = \frac{1}{N}\sqrt{(X_1(i) - \overline{X}(i))^2 + \ldots + (X_N(i) - \overline{X}(i))^2}$$

where $SD(i)$ is the standard deviation for the i^{th} bin, $X_j(i)$ is the j^{th} bundle's average value of the i^{th} bin, and $\overline{X}(i)$ is the average value of the i^{th} bin for the entire data set. Reactive trajectories are tested in bundles until the standard deviation of each bin's data among the bundles is within a given tolerance. Therefore, no more data is computed than is needed to precisely bound the error of the lifetime distribution. Finally, since some trajectories will be expected to stay in the bound region for a very long time, it is possible to obtain a lifetime distribution with a finite number of bins and lump all lingering trajectories into an error-bounded tail.

2c Key Ingredients Needed to Apply Method

A major strength of this method is the generality of systems to which it may be applied. The key features needed by a system to apply this method are: a Rill's region with a saddle separating the bound and unbound states, and a nondegenerate Hamiltonian vector field. Distributions obtained from this method may be combined to synthesize reaction rates in multi-channel systems[5], as long as each channel has a compact intersection with some transverse cut and each reaction state has a clear phase space volume.

3 Method Applied to the Rydberg Scattering Reaction

The Hamiltonian for electron scattering in the Rydberg atom with crossed fields is given by

$$H = \frac{1}{2}\left(p_x^2 + p_y^2 + p_z^2\right) + \frac{1}{2}\left(xp_y - yp_x\right) + \frac{1}{8}\left(x^2 + y^2\right) - \epsilon x - \frac{1}{\sqrt{x^2 + y^2 + z^2}}.$$

A stark saddle point exists at $x = \frac{1}{\sqrt{\epsilon}}, y = 0, z = 0, p_x = 0, p_y = -\frac{1}{2\sqrt{\epsilon}}, p_z = 0$ with energy $E_S = -2\sqrt{\epsilon}$. For energies above E_S, the TS opens and connects the bound and unbound regions of phase space. For consistency with the literature, the all simulations are performed with energy $h = -1.52$ and electric field strength $\epsilon \in (.57765, .60000)^{\dagger}$.

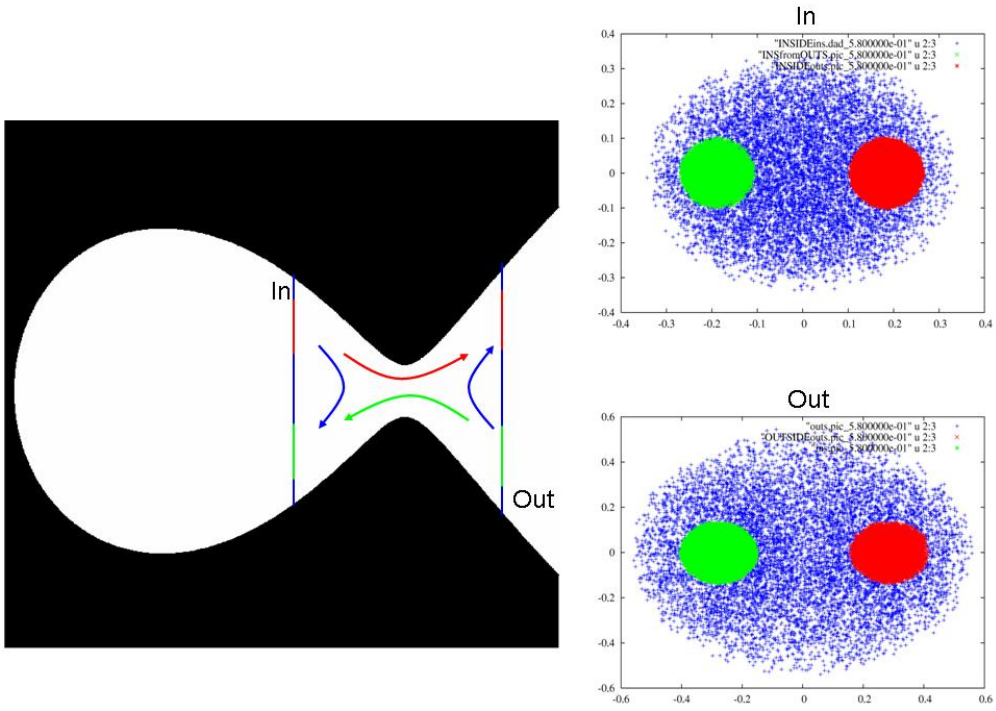

Figure 5: Reactive & unreactive trajectories in the Rydberg scattering reaction. Trajectories are either transit orbits entering (green) and exiting (red) the Hill's region or nontransit orbits (blue). Green points at C_{out} are sent into the bound state, yielding lifetime distribution data. Hill's regions for various ϵ's are found in Appendix A.

In addition to recording the bound time for each trajectory, the number of orbits it makes around the nuclear core is also recorded. To determine the number of nuclear orbits, it is sufficient to have a counter which increases every time the bound trajectory passes a fixed cut transverse to the bound orbits, say at $\Sigma_{y=0}^{x<0}$. To streamline this process, the integrator determines if the trajectory has passed through the cut as it updates the trajectory.

[†]Holding energy fixed and varying ϵ has a similar effect to fixing ϵ and varying energy.

3a Data and Comparison with Literature

Below, the scattering lifetime of electrons in the Rydberg atom is computed using the new method. To start, I plot the Hill's region and Hamiltonian energy surfaces to get an idea of where to place the inside and outside cuts; placing C_{in} and C_{out} a distance of .1 from the fixed point cut C_{fp} works well. Finally, I confirm that the reaction coordinate through the saddle is locally the x-axis (corresponding to the saddle direction).

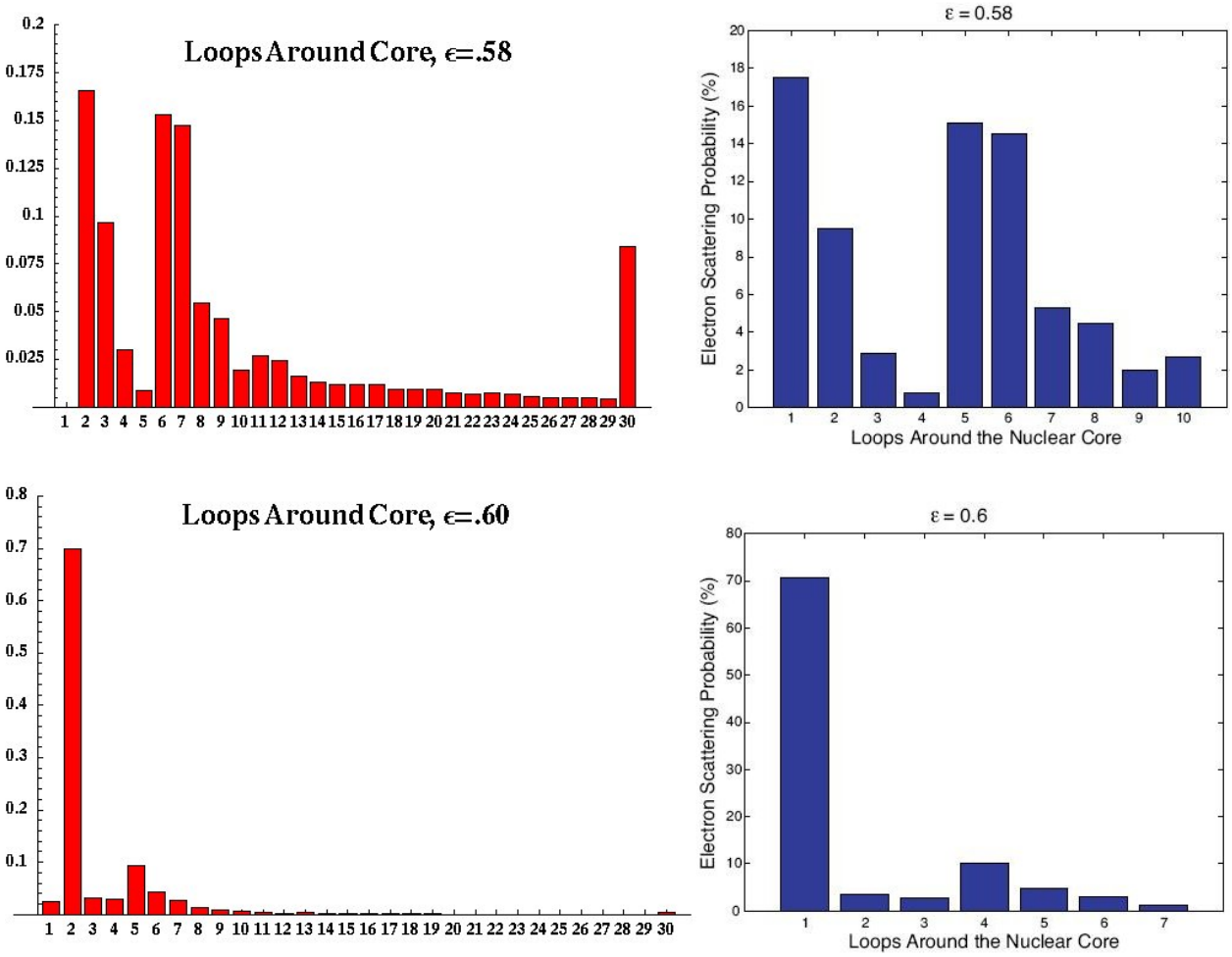

Figure 6: Comparison of Lifetime Distributions computed by this method (left) and the method of Gabern *et al.* (right). Electric field strengths of $\epsilon = .58$ (top) and $\epsilon = .60$ (bottom) are used. This method required 148,000 trajectories at $\epsilon = .58$ and 178,000 trajectories at $\epsilon = .60$ to compute 30 bins, each with error $< .1\%$. The method of Gabern *et al.* uses 1,000,000 points for each ϵ.

The method of Gabern *et al.* uses 1000k points, takes 2 days, and gives no error estimate on the data. The new method requires ∼5k points and 3-5 min to compute data with error $< .5\%$ per bin and ∼150k points and 1 hr to compute data with error $< .1\%$ per bin.

3b Kinetic & Potential Energy Distribution of Entering and Exiting Trajectories

Once a bounding box is obtained, reactive trajectories may be quickly sampled, making it possible to obtain fast-converging distributions for a number of physically relevant quantities in addition to the lifetime distribution. Trajectories in a scattering reaction, for example, are likely to have a different partition of energy into translational and rotational modes before and after the reaction. The total energy is therefore broken into three modes, $K_1 = \frac{1}{2}\left(p_x^2 + p_y^2 + p_z^2\right)$ (translational), $K_2 = \frac{1}{2}\left(xp_y - yp_x\right) + \frac{1}{8}\left(x^2 + y^2\right)$ (rotational), and $V = H - K_1 - K_2$ (potential). The energy partition among these three modes is recorded for a number of reactive trajectories before and after the scattering reaction. This results in distributions for these quantities at the entrance and exit. In addition, it is possible to obtain a distribution for how individual trajectories are expected to shift energy during the reaction by computing ΔK_1, ΔK_2 and ΔV for each trajectory.

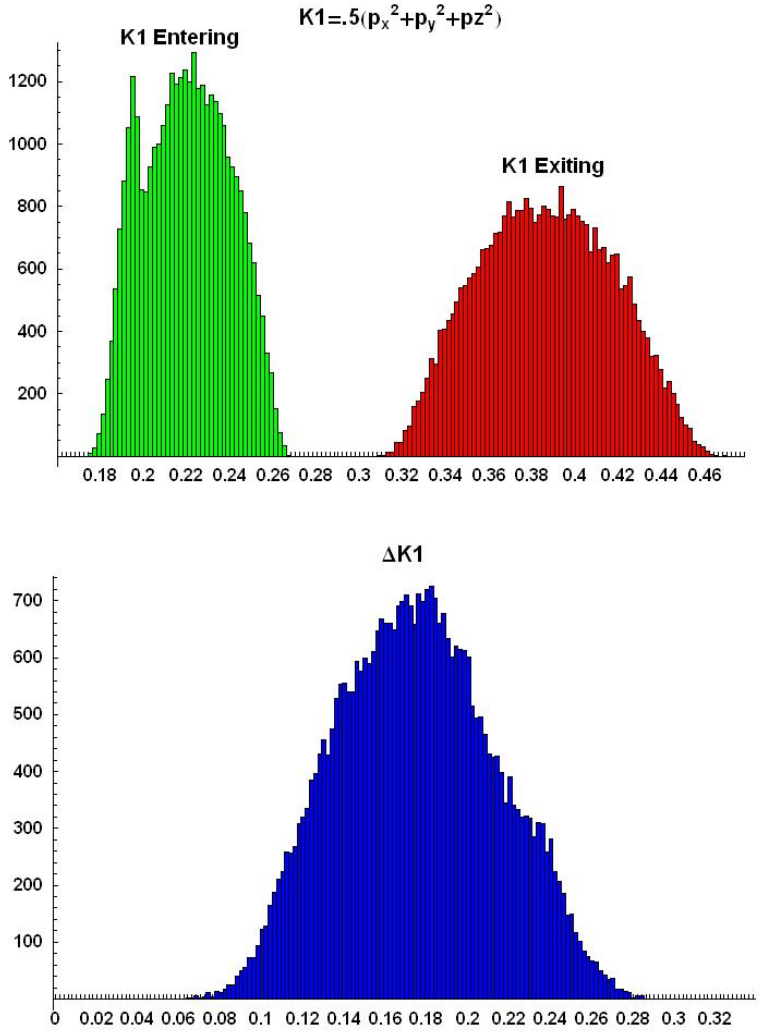

Figure 7: Distribution for $K_1 = \frac{1}{2}\left(p_x^2 + p_y^2 + p_z^2\right)$ of entering (green) and exiting (red) trajectories. ΔK_1 for each trajectory is shown in blue.

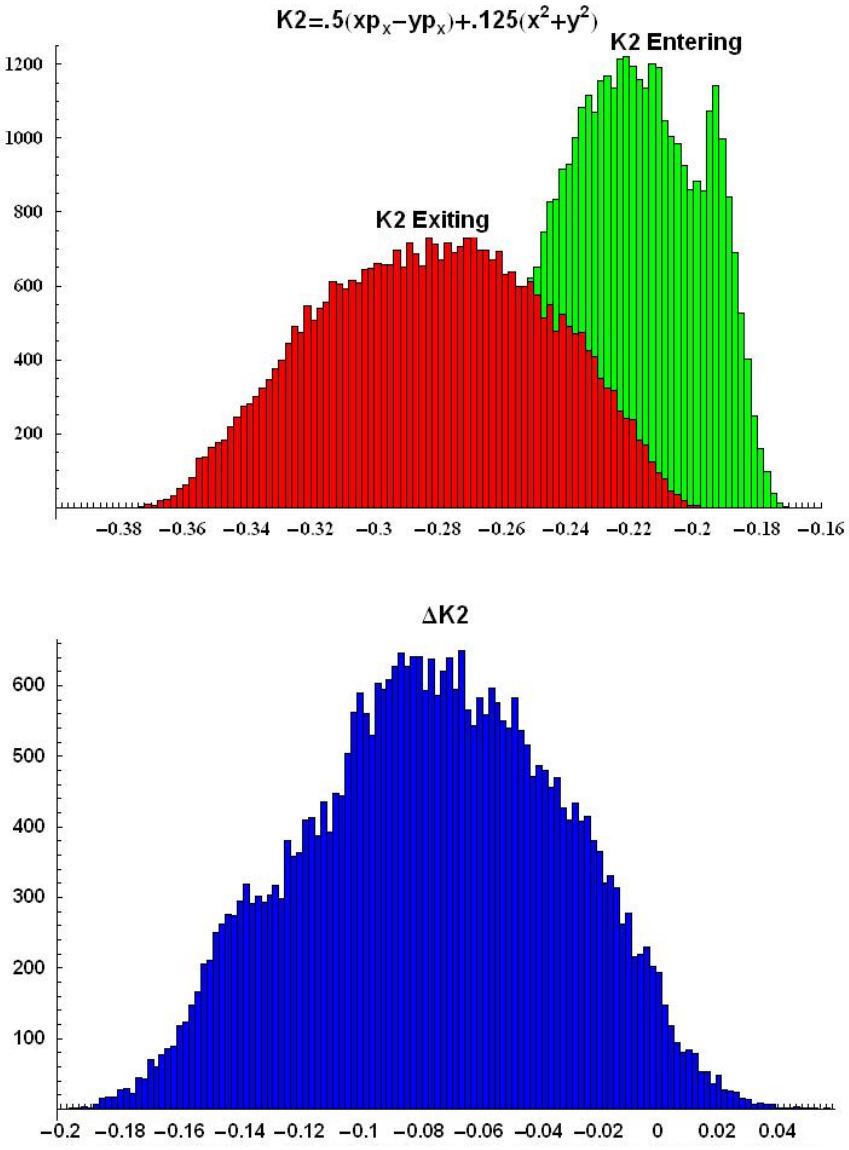

Figure 8: Distribution for $K_2 = \frac{1}{2}(xp_y - yp_x) + \frac{1}{8}(x^2 + y^2)$ of entering (green) and exiting (red) trajectories. ΔK_2 for each trajectory is shown in blue.

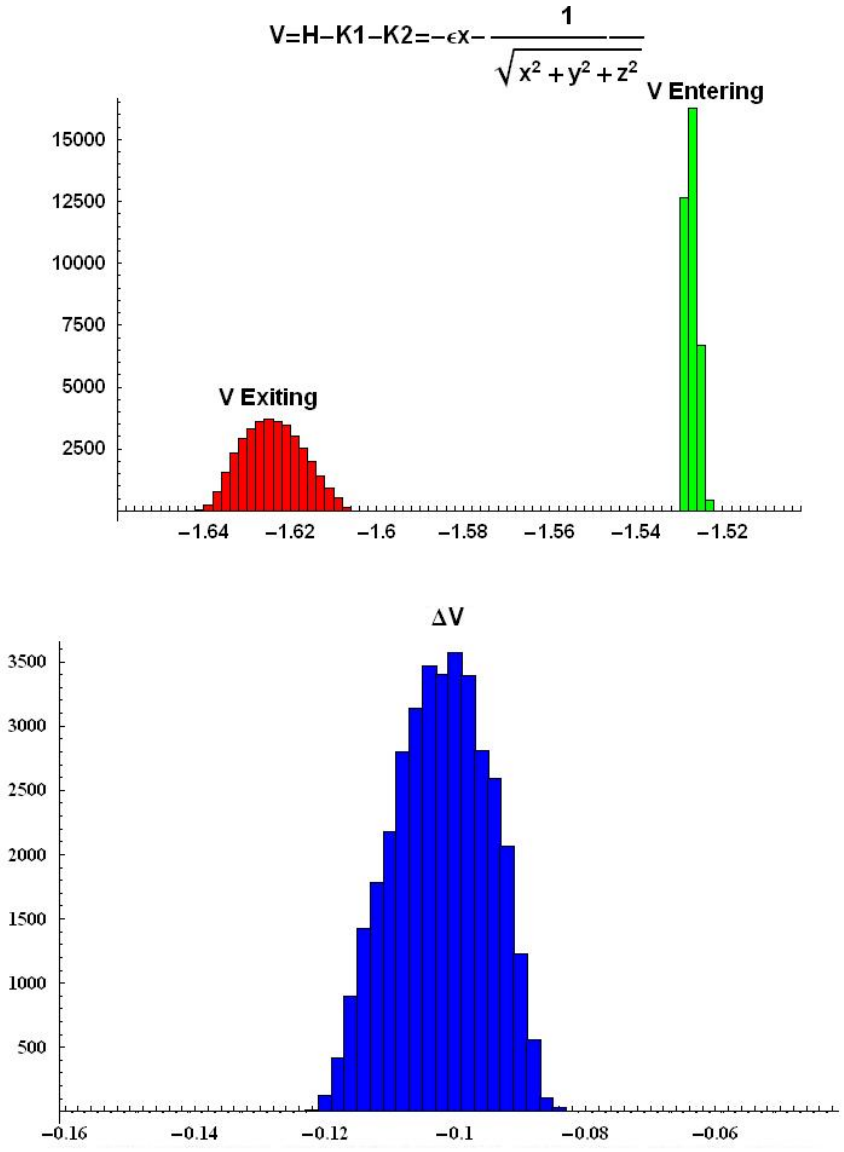

Figure 9: Distribution for $V = H - K_2 - K_1$ of entering (green) and exiting (red) trajectories. ΔV for each trajectory is shown in blue.

From the previous three figures, it may be concluded that after the reaction, trajectories gain translational kinetic energy, losing both rotational and potential energy in the process.

3c Dual Method Test - Lifetime Distribution of Exiting Trajectories Integrated Backward

To further test the method, I construct a dual method which first identifies a box at C_{out} bounding those reactive trajectories which are leaving the bound region, and then integrates the trajectories *backward* along the vector field and back into the bound state. These trajectories are flowed backward along the vector field until they escape into the unbound region. Trajectories that escape in this way correspond to forward reactive trajectories that are on their way into the bound state. Therefore, the dual method takes a reactive trajectory which is exiting the bound region and flows it backward until the instance when the trajectory first enters the bound state. Because a trajectory cannot escape the bound state without first entering, the lifetime distributions obtained using the dual method should precisely match the distribution obtained by regular application of the method. It is clear from the data below that both the method and its dual produce the same lifetime distributions.

The dual method test proves useful when there is no known lifetime distribution to compare results with. For example, a disparity between the two distributions would indicate that either one or both of the methods are not selecting unbiased trajectories, or there is some integration error.

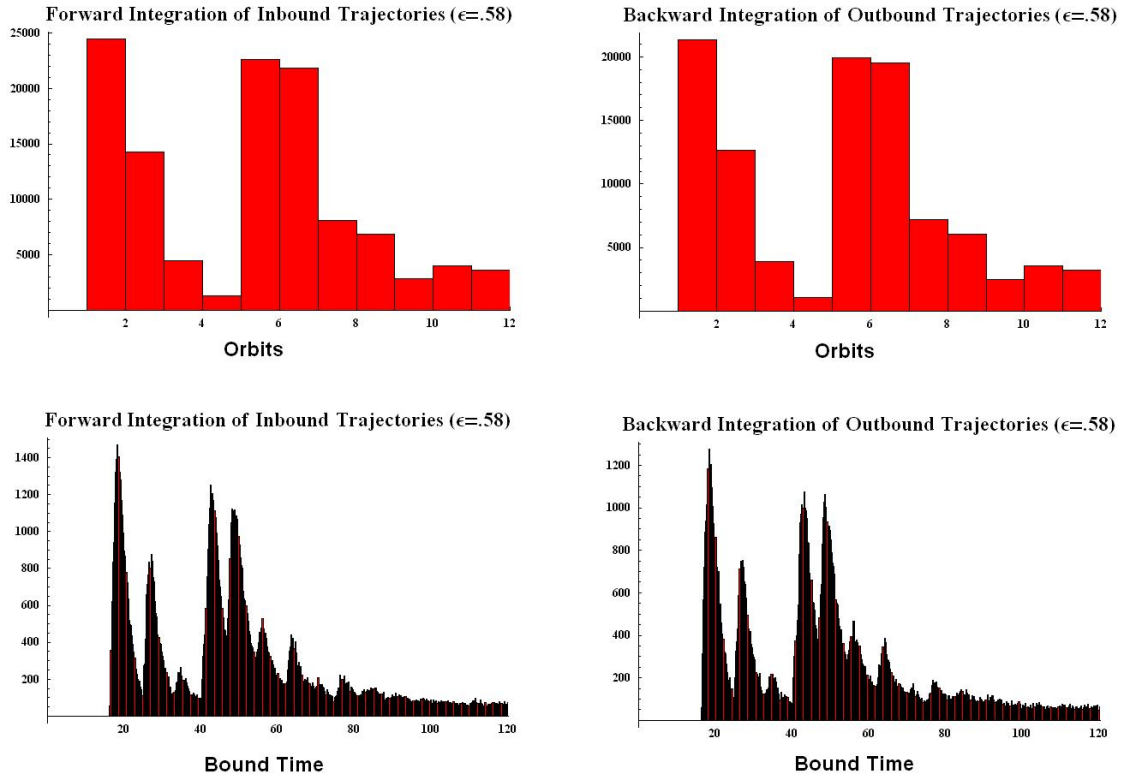

Figure 10: Comparison of Lifetime Distributions at $\epsilon = .58$ computed by integrating inbound trajectories forward (left) and outbound trajectories backward (right) along the vector field. The top two tiles show the number of nuclear core orbits and the bottom tiles show the bound times.

3d Higher DOF Analog of Rydberg Atom

Because of the simplicity of the Rydberg scattering model, it serves as a base for generalizing the method to higher DOF systems. The Rydberg atom has three degrees of freedom, and these may be classified as $x-$like, $y-$like, and $z-$like. Computing transport rates for a generalized Rydberg atom with 4DOF (1 $x-$like variable, 1 $y-$like variable, and 2 $z-$like variables) is theoretically no more difficult than adding the following terms to the Hamiltonian:

$$H(x,y,z,w) = \frac{1}{2}\left(p_x^2 + p_y^2 + p_z^2 + p_w^2\right) + \frac{1}{2}\left(xp_y - yp_x\right) + \frac{1}{8}\left(x^2+y^2\right) - \epsilon x - \frac{1}{\sqrt{x^2+y^2+z^2+w^2}}.$$

Lifetime distributions for 3-8DOF systems (each constructed by adding another $z-$like variable):

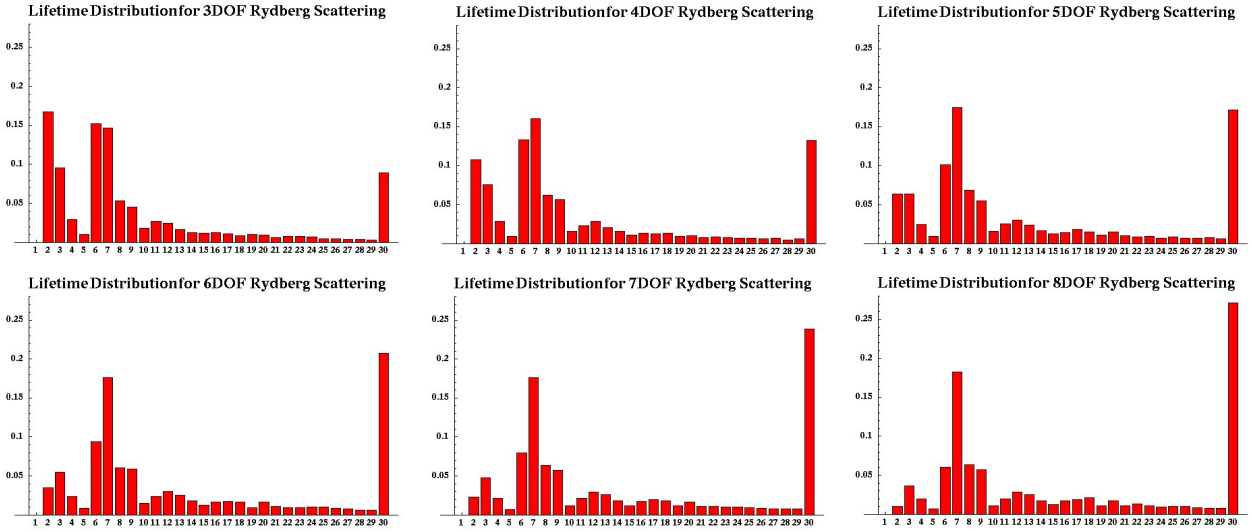

Figure 11: Lifetime Distributions for core loops of higher dimensional Rydberg scattering analogs.

Each of these lifetime distributions was computed in 5-20 minutes, despite the dimensionality of the systems. Because solving for the lifetime distribution is a one dimensional problem, it converges in about the same amount of time for various DOF problems. At 9DOF, however, the bounding box sampling became a computational bottleneck for the computation. It is worth noting that this is a simple make-believe system, and that realistic higher DOF models may be more nonlinearly coupled.

In addition to adding $z-$like variables, it is possible to add another $x-$like degree of freedom, resulting in a rank-2 saddle system. Although the set of reactive trajectories no longer has a compact intersection with a transverse cut near the saddle, the generality of the above method should be well suited for tackling high rank saddles.

3e Experimental Verification and Gaussian Energy Sampling

Because this method was developed to compute accurate rate constants in chemical scattering reactions, it is important to consider how numerical results may be experimentally verified. Sampling points from a fixed energy level set, for example, is not physically realistic, since any experiment will have some expected error for the energy of reactive trajectories. Choosing a Gaussian distribution around a target energy is more physically relevant for experimental verification.

When sampling with a Gaussian energy distribution, it is natural to ask whether or not non-RRKM structure persists. If non-RRKM structure only exists on a fixed energy level set, then the reaction is *effectively RRKM* because the non-RRKM structure is destroyed with the addition of any experimental error. The experiment below samples the Rydberg scattering problem with a Gaussian energy distribution around the target energy $h = -1.5$ (in section 3, this would correspond roughly to $h = -1.52$ and $\epsilon = .60$). Notice that the lifetime distribution remains non-RRKM.

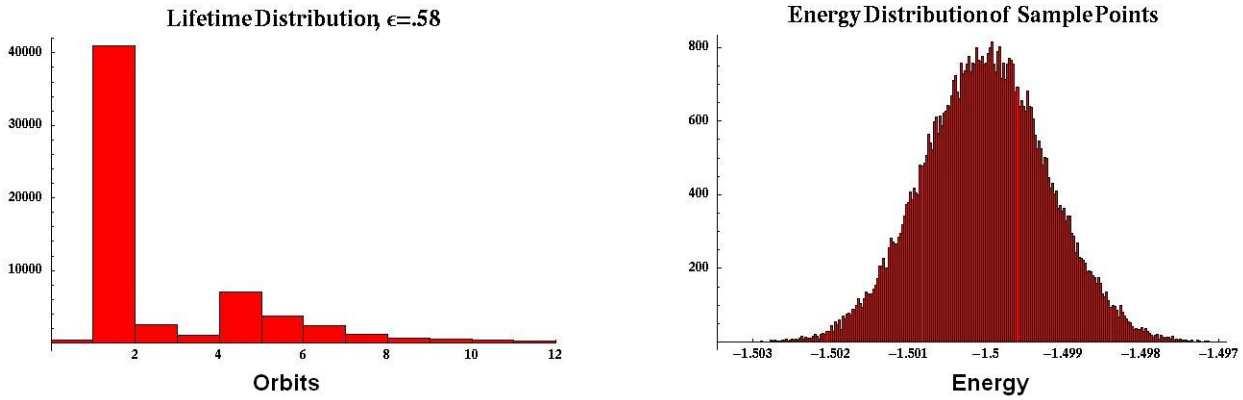

Figure 12: Lifetime Distribution for fixed $\epsilon = .58$ and a sample with a Gaussian energy distribution.

4 Planar, Nonzero Angular Momentum Scattering of H_2O with H_2

4a Model Hamiltonian and Reduction

The model for the planar scattering of H_2O-H_2 used by Wiesenfeld *et al.* is given by the 4 DOF system shown in figure 13. The intermolecular radius R may be thought of as the reaction coordinate along which the scattering reaction takes place, and the angles χ_α and χ_β are the orientations of the H_2O and H_2 molecules with respect to the axis between them. The fourth variable θ is the angle between the intermolecular axis and the laboratory frame. The Hamiltonian uses variables $R, \alpha = \chi_\alpha, \beta = \chi_\beta - \chi_\alpha$ and is given by

$$H = \frac{p_R^2}{2m} + \frac{(p_\theta - p_\alpha)^2}{2mR^2} + \frac{(p_\alpha - p_\beta)^2}{2I_a} + \frac{p_\beta^2}{2I_b} + V$$

where V is the potential energy function obtained by summing dipole/quadrupole, dispersion, induction, and Leonard-Jones potentials. Since the potential is invariant under rotations of the system in the laboratory frame, it is possible to reduce the system from 4DOF to 3DOF by simply considering the Hamiltonian in the rotating frame. The total angular momentum p_θ is a conserved quantity and is therefore fixed to a specific $p_\theta \equiv J$. In this way, the total angular momentum J is a *parameter* for the reduced system. Reduction from 4DOF to 3DOF provides two immediate benefits: first, lower DOF systems are much easier to work with both from a computational and a visualization standpoint. Second, in reduced coordinates, the relative TS becomes an actual TS located near a saddle of the reduced potential energy surface. The details of this reduction in the general context of abelian reduction[9,10] are presented in Appendix C.

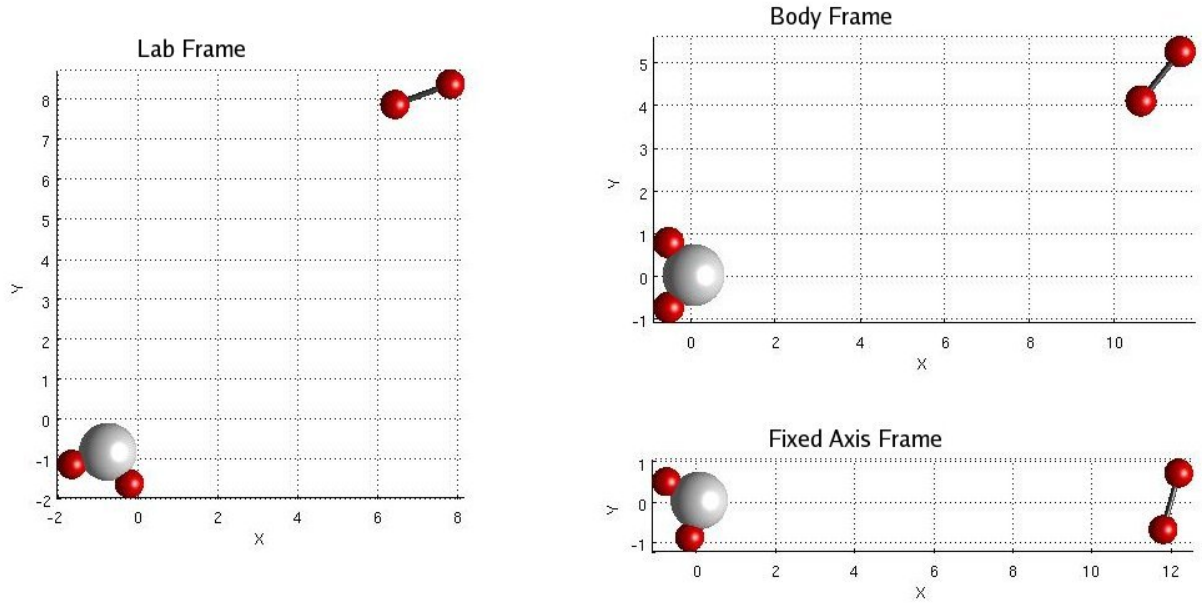

Figure 13: Schematic for reduction to rotating frame. See Appendix C for details.

The reduced system has four saddle points. Two of the saddles are rank-1 which means that they have the desired energy spectrum saddle$\times$center$\times$center. For energies slightly larger than that of the lowest energy saddle, a TS opens up connecting the bound and unbound states.

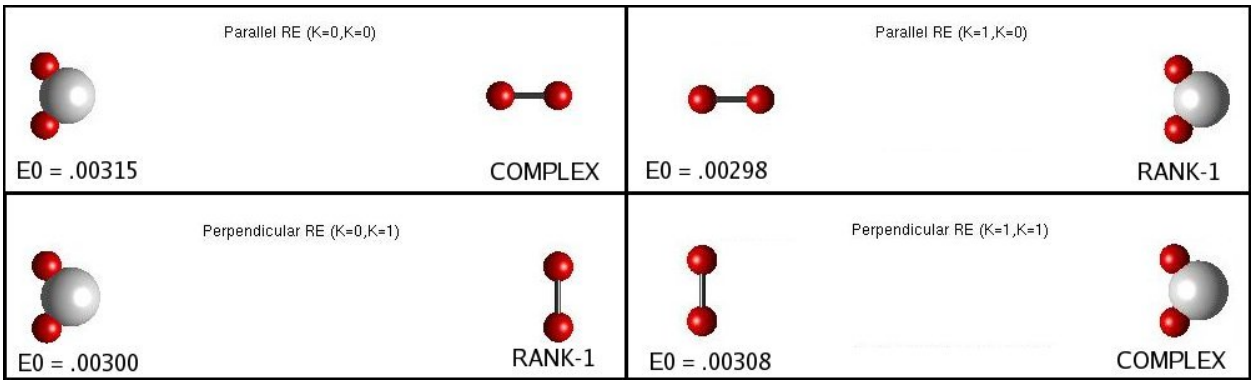

Figure 14: Four Saddle Points Exist for Reduced System.

The two regions are separated in configuration space by a Hill's region, which is simply the projection of the energy surface specified by $H = E_0$ onto configuration space. This projection indicates which configurations are energetically forbidden, and is characterized by a single opening (located at energies slightly above the TS). Therefore, all reactive trajectories entering the bound state must pass through the opening of the Hill's region. It can be seen in figure 15 that for energies much larger than the TS, more scattering channels associated with the other three saddles open up. It is worth noting that the Hill's region is viewed in physical coordinates $(x, y, \beta) = (R\cos(\alpha), R\cos(\beta), \beta)$ where $(R, \alpha, \beta) = (R, \chi_\alpha, \chi_\beta - \chi_\alpha)$ are body-frame coordinates obtained by reducing the system to the coordinate frame which is fixed to and aligned with the water molecule. A three dimensional Hill's region may be visualized by stacking $\beta-$slices; see Appendix B.

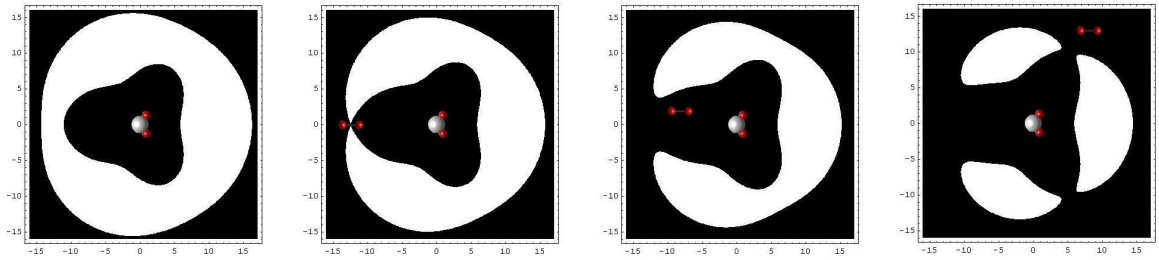

Figure 15: Hill's region at various energies of H_2O-H_2 potential. The white region represents energetically forbidden configurations on the slice $\beta = 0$ (corresponding to H_2 aligning with H_2O dipole). (far left) $E_0 < E_S$. (left) $E_0 = E_S$. (right) $E_0 > E_S$. (far right) $E_0 >> E_S$.

4b Potential Pitfall of Model - Collisions

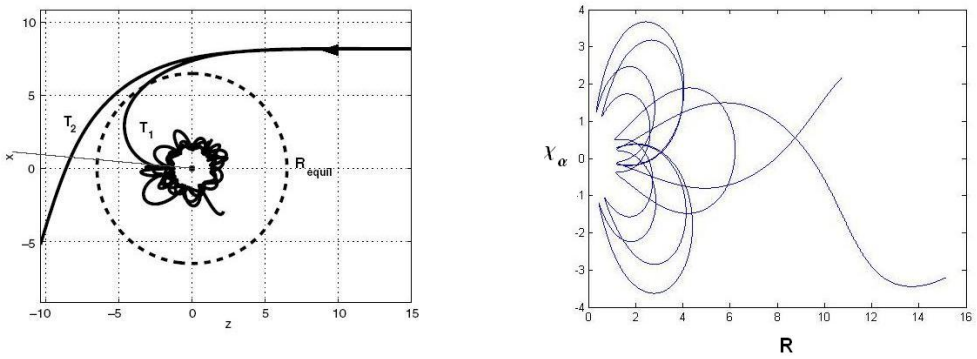

Figure 16: H_2-H_2O Collisions are typical for this potential. (left) A diagram shown in Wiesenfeld *et al.*, (right) A trajectory computed in body frame coordinates.

Despite providing an illustration for how to reduce a nonzero angular momentum scattering model in order to identify the saddle underlying a TS, the scattering model for H_2O-H_2 has a number of potential defects. Instead of reactive trajectories making orbits around the scattering core (H_2O), the hydrogen molecule repeatedly collides with the water molecule. Because the products of a scattering reaction do not change molecular combination, the collisions of the hydrogen molecule with the hydrogens on the water raises suspicion that the system is actually undergoing a non-scattering chemical exchange reaction. Because of the Leonard-Jones potential, these collisions are modeled as elastic, when in reality the conditions may be right for chemical bonds to break. In addition to the physical implications of molecular collisions in a scattering reaction, these collisions are numerically volatile, slowly driving the trajectory off of the relevant energy surface. Finally, the depth of the potential well results in a large number of average collisions, making runtimes intractable for a majority of trajectories at low energies.

These difficulties not withstanding, the scattering model still possesses all of the ingredients needed by the above bounding box and sampling methods. In fact, it may be thought of as a particularly poorly behaved model system, from which the method is still able to extract structured data.

4c Data and Analysis

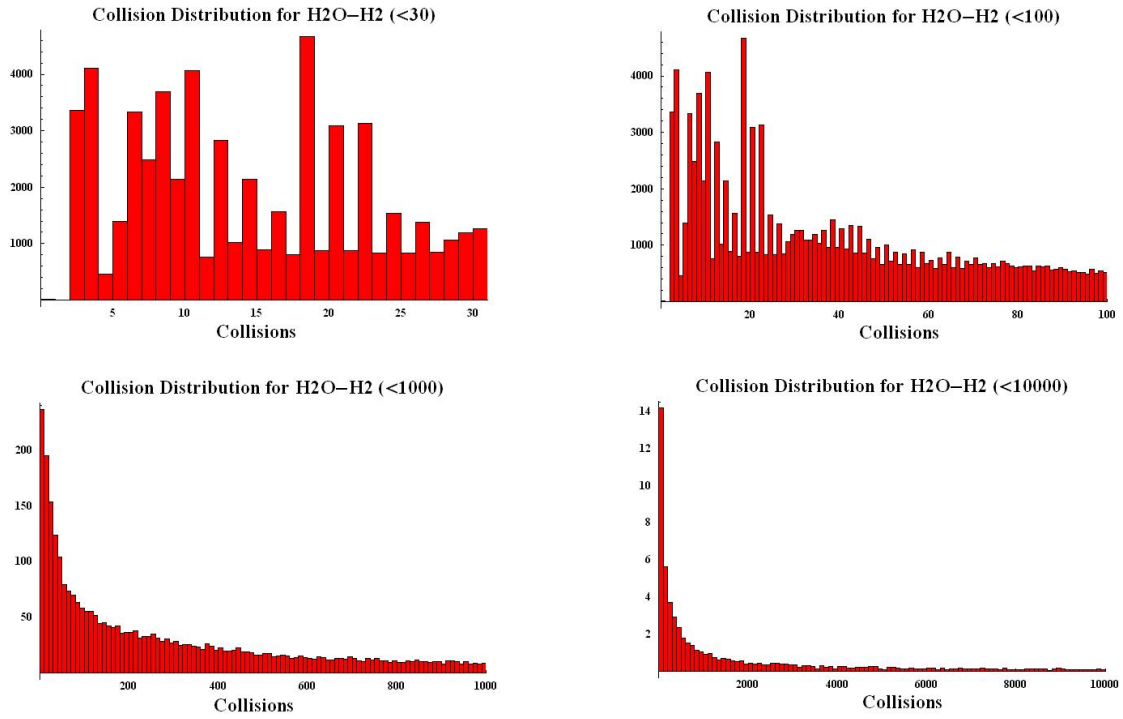

Figure 17: Lifetime Distribution for the planar H_2O-H_2 scattering reaction for various bin sizes and maximum recorded collisions. Data takes $\sim$2 days to compute.

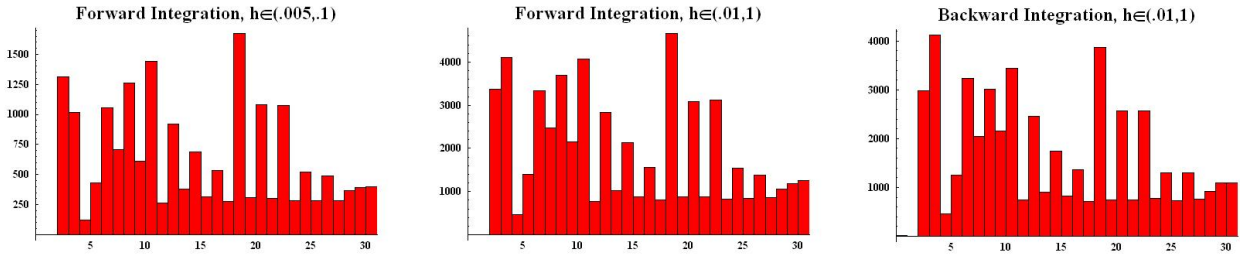

Figure 18: Lifetime Distributions using forward (middle) and backward (right) integration are compared. LD is also computed using a smaller time step h (left).

The above data was computed for the H_2O-H_2 scattering potential with energy H =3.e-3 and total angular momentum $J = 7.6193$. It is clear that on a large enough scale, the lifetime distribution is strongly statistical. Below $\sim$100 collisions, however, there is definite non-RRKM structure which is displayed by the above data. These trajectories account for only a small fraction of total reactive trajectories, yet they illustrate that even in a strongly statistical distribution, there may still be some types of collisions that are favored locally over others. In fact, one could test any number of hypotheses such as *trajectories are more likely to have an even number of collisions before escaping.*

5 Conclusions and Future Directions

The method developed here is applied to compute accurate lifetime distributions for two 3DOF scattering reactions. The lifetime distribution for the Rydberg scattering matches the distribution of Gabern *et al.*, confirming the accuracy of the method. Additionally, the method converges upon the lifetime distribution very quickly, on the order of an hour. Although the H_2O-H_2 model is numerically more involved, the method is able to compute a lifetime distribution on various scales with about 2 days of computation. We also address the introduction of nonzero angular momentum into scattering reactions by reducing the dynamics to a $J-$level set of the total angular momentum and identifying the TS near a rank-1 saddle of the reduced system.

The next step in order to demonstrate the power and scope of this new method is to apply it to study a $>$ 3DOF chemical system. In addition to obtaining a lifetime distribution, it will be important to investigate experimental verification of the data to broaden the range of applicability of these methods. An ideal system to expand this analysis to is the generalized 9DOF H_2O-H_2 scattering model recently developed by Wiesenfeld *et al.*[2,3].

5a Improving the Method

To streamline the method for $\geq$ 3DOF systems, there are two main avenues for improvement. First, implementing a tighter fitting bounding box, perhaps by utilizing a patchwork of smaller boxes, will greatly improve the efficiency of stage 2.) sampling for reactive trajectories.

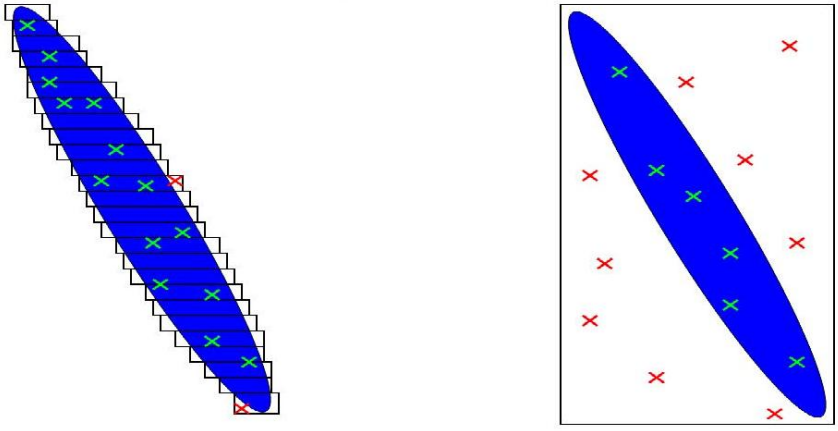

Figure 19: Improved Bounding Box which uses several smaller boxes to obtain a tighter fit (left), and Current Bounding Box (right).

The second direction to pursue in order to improve the method is the introduction of a variational integrator to accurately compute reactive trajectories. Variational integrators not only keep trajectories on the energy surface, which is ideal for long integrations involving sharp collisions, but also allows for larger time steps which translates to faster run times. Since the bulk of the com-

putation is currently involved in integration of reactive trajectories, stage 3.), this is an ideal setting to showcase the power and speed of variational integration. Finally, one can see the numerical error in using a Runga-Kutta 78 integrator in figure 17, where halving the time step or applying the dual method test both result in subtly different lifetime distributions.

5b Physically Meaningful Distributions

Although the above method has been designed to quickly compute accurate lifetime distributions for rank-1 transport phenomena, the bounding box/random sampling tools allow for the fast computation of distributions for nearly any property of a trajectory that can be quantified. Finding clever physical quantities to compute distributions of, may prove useful in analyzing physical systems such as chemical reactions or asteroid capture dynamics.

5c Experimental Verification and Half Scattering Reactions

Experiments that involve scattering reactions typically study what is known as a half scattering reaction. Unlike the full scattering reaction, where trajectories start and end in the unbound state, the reactive trajectories in a half scattering reaction start in the bound state and end in the unbound state. To obtain experimental verification of the results produced with the above method, it will be necessary to add half scattering functionality to the lifetime distribution code. This involves determining a box bounding reactive trajectories *inside* the bound state, transverse to the bundle of reactive trajectories which are found in the bound state. A good candidate for such a cut is $\Sigma_{y=0}^{x<0}$ in the Rydberg scattering reaction.

5d Color Coding Trajectories at the Bounding Box and Poincaré Cut

To determine the lifetime distribution for a scattering reaction, a large number of reactive trajectories are sampled at random from the bounding box. These trajectories are sent into the bound state and every collision/orbit is recorded. The records of how many collisions/orbits each trajectory makes is stored with the initial condition from the bounding box, and it is a trivial step to also record the multiple intersections of each individual trajectory with the Poincaré cut $\Sigma_{y=0}^{x<0}$. Assigning each number of collisions/orbits a color, it is possible to color code the phase space regions in the bounding box and at the Poincaré cut containing all reactive trajectories. This would result in a color coded *heat map* indicating regions of greater and lesser scattering activity, which would couple the reactive trajectories at the bounding box to their intersections at the Poincaré Cut. In a similar effort, it would be possible to continuously deform initial conditions corresponding to $m-$loops into initial conditions corresponding to $n-$loops, perhaps with intermediate $k-$loops.

Appendix A Rydberg Scattering Reaction - Energy Surfaces

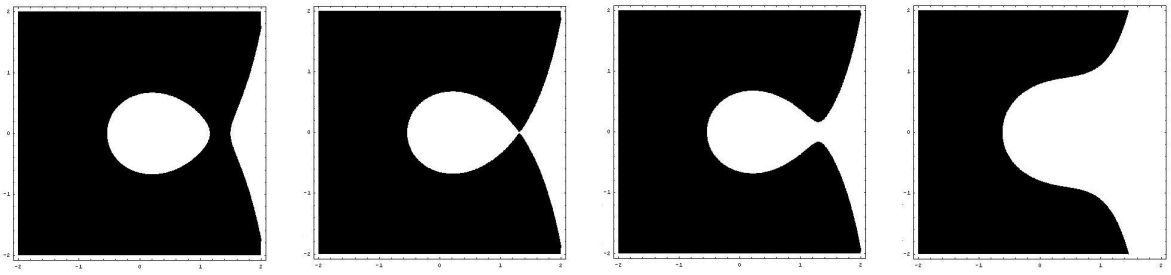

Figure 20: Hill's region for Rydberg scattering reaction at various energies. The white region represents energetically forbidden configurations on slice $z = 0$. (far left) $E_0 < E_{TS}$. (left) $E_0 = E_{TS}$. (right) $E_0 > E_{TS}$. (far right) $E_0 >> E_{TS}$.

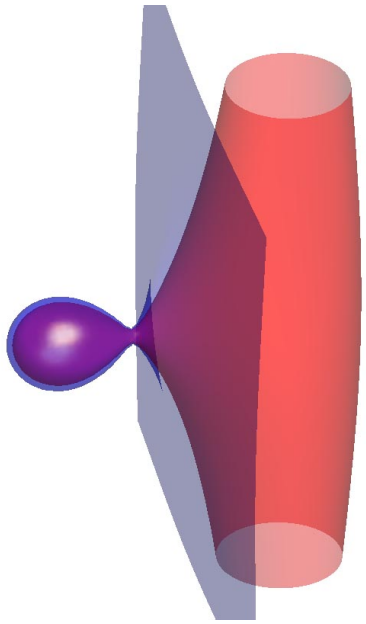

Figure 21: Hill's region at $\epsilon = .58$ (blue) and Hamiltonian energy surface with momentum restricted to fixed point momentum (red).

Appendix B Planar H_2O-H_2 Scattering Reaction - Energy Surfaces

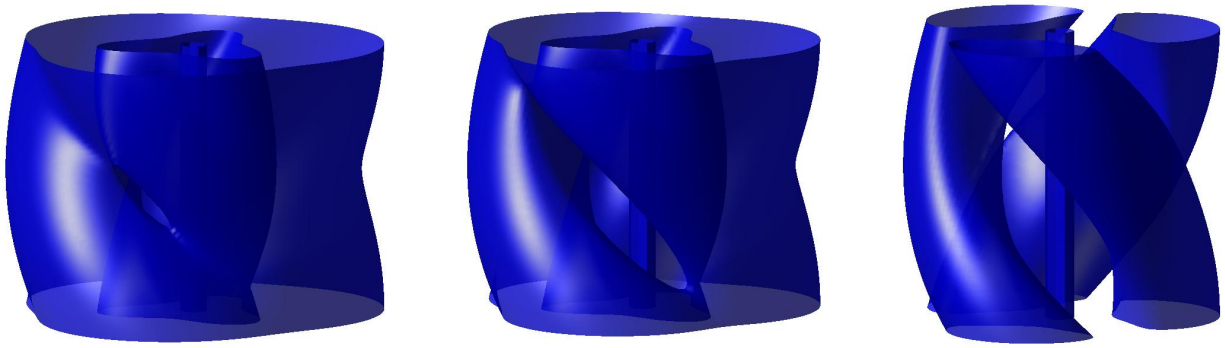

Figure 22: 3D Hill's region constructed by stacking $\beta-$slices for various energies.

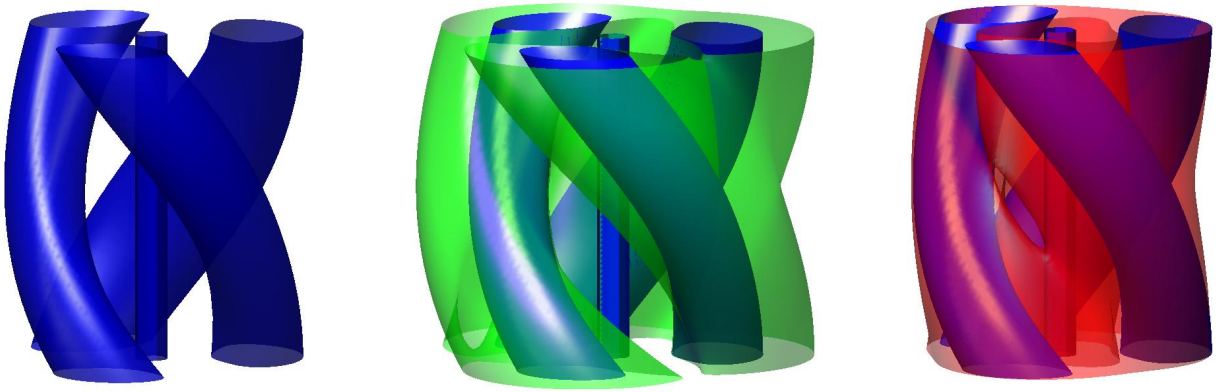

Figure 23: Blue Hill's region for H_2O-H_2 scattering reaction (left), superimposed with green Hamiltonian surface for momentum of entering trajectory (middle), and superimposed with red Hamiltonian surface for momentum of exiting trajectory (right).

Appendix C Planar H_2O-H_2 Scattering Reaction - Abelian Reduction

The general method of abelian reduction[9,10] is applied to the rigid rotor model for the scattering of H_2-O-H_2.

C1 Equations of Motion

We consider a system comprised of two planar molecules H_2-O and H_2, approximated as rigid rotors, interacting in the plane via the approximate potential of Wiesenfeld *et al*. Configuration space for this system is SE(2)×SE(2), determined by an (x, y) position and orientation ϕ for each body. Reduction via translational symmetry, i.e. conservation of linear momentum, simplifies the configuration space to SE(2)×SO(2)≅ SE(2)× S^1. The system is also invariant under rotations in the plane, as seen by the $\theta-$invariance of the potential of Wiesenfeld *et al*. Reduction of these planar rotations further simplifies the configuration space to SE(2), which is expressed in body-frame coordinates.

Fixing the origin at the center of mass of the H_2O molecule in an inertial frame that translates with the center of mass of both bodies, we may define the Kinetic Energy as follows

$$T(\dot{\mathbf{q}}, \dot{\phi}_a, \dot{\phi}_b) = \frac{m}{2}\|\dot{\mathbf{q}}\|^2 + \frac{I_a}{2}\dot{\phi}_a^2 + \frac{I_b}{2}\dot{\phi}_b^2 \tag{1}$$

The conjugate momenta are found via the Legendre transformation (with conservative potential energy): $\mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}} = m\dot{\mathbf{q}}$, $j_i = \frac{\partial L}{\partial \dot{\phi}_i} = I_i\dot{\phi}_i$. This yields the following Hamiltonian:

$$H = \frac{\|\mathbf{p}\|^2}{2m} + \frac{j_a^2}{2I_a} + \frac{j_b^2}{2I_b} + V(\mathbf{q}, \phi_a, \phi_b) \tag{2}$$

The potential V, given below, does not depend on θ, and we seek to reduce via this rotational symmetry.

$$\begin{aligned} V(R, \chi_a, \chi_b) =& \frac{3\mu Q}{4R^4}\left\{\cos\chi_a\left(3\cos^2\chi_b - 1\right) - \sin\chi_a \sin 2\chi_b\right\} \\ &- \frac{1}{R^6}\left\{3(A-B)\cos^2\chi_b + (A+5B)\right\} \\ &- \frac{\mu^2\bar{\alpha}\left(3\cos^2\chi_a + 1\right)}{2R^6} - \frac{\mu^2\delta\alpha}{4R^6}\left\{\cos 2\chi_b\left(3\cos^2\chi_a + \cos 2\chi_a\right) + 2\sin 2\chi_a \sin 2\chi_b\right\} \\ &+ \frac{C_{\mathrm{LJ}}}{R^{12}} \end{aligned}$$

Notice that the potential is written in terms of relative angles $\chi_a = \phi_a - \theta$ and $\chi_b = \phi_b - \theta$.

C2 Trivializing Rotational Symmetry

We first apply two canonical coordinate transformations that trivialize the action of the symmetry group $G = S^1$ on the configuration space. The first change is the introduction of polar coordinates

$$q_x = R\cos\theta, \qquad\qquad q_y = R\sin\theta$$

which makes clear the fact that θ is the angle to be reduced. The second change is the introduction of relative coordinates:

$$\begin{aligned} \chi_a &= \phi_a - \theta, & p_a &= j_a \\ \chi_b &= \phi_b - \theta, & p_b &= j_b \end{aligned}$$

We may now reformulate the Kinetic Energy in terms of these new coordinates:

$$T(\dot{R},\dot{\theta},\dot{\chi}_a,\dot{\chi}_b) = \frac{m}{2}\dot{R}^2 + \frac{mR^2}{2}\dot{\theta}^2 + \frac{I_a}{2}\left(\dot{\chi}_a+\dot{\theta}\right)^2 + \frac{I_b}{2}\left(\dot{\chi}_b+\dot{\theta}\right)^2 \tag{3}$$

$$= \frac{m}{2}\dot{R}^2 + \frac{mR^2+I_a+I_b}{2}\dot{\theta}^2 + \left(I_a\dot{\chi}_a + I_b\dot{\chi}_b\right)\dot{\theta} + \frac{I_a}{2}\dot{\chi}_a^2 + \frac{I_b}{2}\dot{\chi}_b^2 \tag{4}$$

We therefore have a Lagrangian in the following form:

$$L(q^\alpha,\dot{\theta},\dot{q}^\alpha) = \frac{1}{2}g_{00}\dot{\theta}^2 + g_{0\alpha}\dot{\theta}\dot{q}^\alpha + \frac{1}{2}g_{\alpha\beta}\dot{q}^\alpha\dot{q}^\beta - V(q^\alpha) \tag{5}$$

where g_{ij} is the Kinetic Energy metric,

$$g = \begin{pmatrix} & \dot{\theta} & \dot{R} & \dot{\chi}_a & \dot{\chi}_b \\ \dot{\theta} & mR^2+I_a+I_b & 0 & I_a & I_b \\ \dot{R} & 0 & m & 0 & 0 \\ \dot{\chi}_a & I_a & 0 & I_a & 0 \\ \dot{\chi}_b & I_b & 0 & 0 & I_b \end{pmatrix} \tag{6}$$

and the indices are arranged as (θ, q^α) where $q^\alpha = (R,\chi_a,\chi_b)$. θ is a cyclic variable and its conjugate momentum

$$\begin{aligned} p_\theta = \frac{\partial L}{\partial\dot{\theta}} &= g_{00}\dot{\theta} + g_{0\alpha}\dot{q}^\alpha \\ &= \left(mR^2+I_a+I_b\right)\dot{\theta} + I_a\dot{\chi}_a + I_b\dot{\chi}_b \\ &= l + p_a + p_b \equiv J \end{aligned}$$

is a conserved quantity, namely the total angular momentum.

We may again calculate the conjugate momenta to the variables $q^\alpha = (R,\chi_a,\chi_b)$ via the Legendre transformation: $p_R = \frac{\partial T}{\partial\dot{R}} = m\dot{R}$, $p_a = \frac{\partial T}{\partial\dot{\chi}_a} = I_a\left(\dot{\chi}_a+\dot{\theta}\right) = j_a$, $p_b = \frac{\partial T}{\partial\dot{\chi}_b} = I_b\left(\dot{\chi}_b+\dot{\theta}\right) =$

j_b. In these coordinates the Hamiltonian is $H = \frac{1}{2}g^{ij}p_ip_j + V(R, \chi_a, \chi_b)$ where g^{ij} is the inverse of g_{ij}:

$$H = \frac{p_R^2}{2m} + \frac{(p_\theta - p_a - p_b)^2}{2mR^2} + \frac{p_a^2}{2I_a} + \frac{p_b^2}{2I_b} + V(R, \chi_a, \chi_b) \tag{7}$$

C3 Body Frame

For body frame coordinates we introduce $(R, \alpha, \beta) \equiv (R, \chi_a, \chi_b - \chi_a)$. The kinetic energy may be written as

$$T(\dot{R}, \dot{\theta}, \dot{\alpha}, \dot{\beta}) = \frac{m}{2}\dot{R}^2 + \frac{I_{zz}}{2}\dot{\theta}^2 + (I_a\dot{\alpha} + I_b(\dot{\alpha} + \dot{\beta}))\dot{\theta} + \frac{I_a}{2}\dot{\alpha}^2 + \frac{I_b}{2}(\dot{\beta} + \dot{\alpha})^2$$

with an associated metric tensor

$$g = \begin{pmatrix} mR^2 + I_a + I_b & 0 & I_a + I_b & I_b \\ 0 & m & 0 & 0 \\ I_a + I_b & 0 & I_a + I_b & I_b \\ I_b & 0 & I_b & I_b \end{pmatrix}$$

The corresponding Hamiltonian $H = \frac{1}{2}g^{ij}p_ip_j + V$ is

$$H = \frac{(p_\theta - p_\alpha)^2}{2mR^2} + \frac{(p_\alpha - p_\beta)^2}{2I_a} + \frac{p_\beta^2}{2I_b} + \frac{p_R^2}{2m} + V \tag{8}$$

with conjugate momenta $p_\alpha = p_a + p_b$ and $p_\beta = p_b$.

C4 Body frame - SE(2)

Similarly, we write the kinetic energy in cartesian bodyframe coordinates:

$$T = \frac{m}{2}\left(\frac{x\dot{x} + y\dot{y}}{x^2 + y^2}\right)^2 + \frac{I_{zz}}{2}\dot{\theta}^2 + \left(\frac{I_a + I_b}{x^2 + y^2}(x\dot{y} - y\dot{x}) + I_b\dot{\beta}\right)\dot{\theta} + \frac{I_a}{2}\left(\frac{x\dot{y} - y\dot{x}}{x^2 + y^2}\right)^2 + \frac{I_b}{2}\left(\frac{x\dot{y} - y\dot{x}}{x^2 + y^2} + \dot{\beta}\right)^2$$

The kinetic energy metric in $\dot{\theta}, \dot{x}, \dot{y}, \dot{R}$ is

$$\begin{pmatrix} I_a + I_b + m(x^2 + y^2) & -\frac{(I_a + I_b)y}{x^2 + y^2} & \frac{(I_a + I_b)x}{x^2 + y^2} & I_b \\ -\frac{(I_a + I_b)y}{x^2 + y^2} & \frac{(I_a + I_b)y^2 + mx^2}{(x^2 + y^2)^2} & \frac{(-(I_a + I_b) + m)xy}{(x^2 + y^2)^2} & -\frac{I_b y}{x^2 + y^2} \\ \frac{(I_a + I_b)x}{x^2 + y^2} & \frac{(-(I_a + I_b) + m)xy}{(x^2 + y^2)^2} & \frac{(I_a + I_b)x^2 + my^2}{(x^2 + y^2)^2} & \frac{I_b x}{x^2 + y^2} \\ I_b & -\frac{I_b y}{x^2 + y^2} & \frac{I_b x}{x^2 + y^2} & I_b \end{pmatrix}$$

The corresponding Hamiltonian is

$$H = \frac{(p_\beta - p_y x + p_x y)^2}{2I_a} + \frac{p_\beta^2}{2I_b} + \frac{p_\theta^2}{2m(x^2 + y^2)} + \frac{p_\theta(yp_x - xp_y)}{m(x^2 + y^2)} + \frac{p_x^2(x^4 + y^2(x^2 + 1))}{2m(x^2 + y^2)} \tag{9}$$

$$+ \frac{p_y^2(y^4 + x^2(y^2 + 1)}{2m(x^2 + y^2)} + \frac{p_x p_y(x^3 y - xy + xy^3)}{m(x^2 + y^2)} \tag{10}$$

C5 Reduced Hamiltonian

Because the system is $\theta-$invariant, to reduce out this cyclic variable amounts to fixing each instance of p_θ equal to γ. This means that in any given instance, we are working on a $\gamma-$level set of the total angular momentum. The details of this process are explained and worked out in detail in C8-9.

The reduced Hamiltonian(s) are as follows:

$$H = \frac{p_R^2}{2m} + \frac{(\gamma - p_a - p_b)^2}{2mR^2} + \frac{p_a^2}{2I_a} + \frac{p_b^2}{2I_b} + V \tag{11}$$

$$H = \frac{(\gamma - p_\alpha)^2}{2mR^2} + \frac{(p_\alpha - p_\beta)^2}{2I_a} + \frac{p_\beta^2}{2I_b} + \frac{p_R^2}{2m} + V \tag{12}$$

As a sanity check, confirm that Eqs. 11 & 12 are equivalent:

Eq. 7
$$\begin{aligned} H =& \frac{p_R^2}{2m} + \frac{(\gamma - p_a - p_b)^2}{2mR^2} + \frac{p_a^2}{2I_a} + \frac{p_b^2}{2I_b} + V \\ =& \frac{p_R^2}{2m} + \frac{p_a^2}{2I_a} + \frac{p_b^2}{2I_b} + \frac{\gamma^2}{2mR^2} - \frac{\gamma(p_a + p_b)}{mR^2} + \frac{(p_a + p_b)^2}{2mR^2} + V \end{aligned}$$

Eq. 13
$$\begin{aligned} H =& \frac{(I_a + I_b)\gamma^2}{(2mR^2)(I_a + I_b + mR^2)} - \frac{(p_a + p_b)\gamma}{mR^2} + \frac{(p_a + p_b)^2}{2mR^2} + \frac{R^2(I_b m p_a^2 + I_a m p_b^2 + I_a I_b p_R^2)}{2mI_a I_b R^2} \\ &+ \frac{\gamma^2}{2(mR^2 + I_a + I_b)} + V \\ =& \frac{p_a^2}{2I_a} + \frac{p_b^2}{2I_b} + \frac{p_R^2}{2m} - \frac{\gamma(p_a + p_b)}{mR^2} + \frac{(p_a + p_b)^2}{2mR^2} + \frac{(I_a + I_b + mR^2)\gamma^2}{(2mR^2)(I_a + I_b + mR^2)} + V \\ =& \frac{p_R^2}{2m} + \frac{p_a^2}{2I_a} + \frac{p_b^2}{2I_b} + \frac{\gamma^2}{2mR^2} - \frac{\gamma(p_a + p_b)}{mR^2} + \frac{(p_a + p_b)^2}{2mR^2} + V \end{aligned}$$

Clearly Eqs. 11 & 12 are equivalent using the relationship $p_\alpha = p_a + p_b$ and $p_\beta = p_b$.

C6 Canonical Transformations

Now, to check that Eq. 7 is the canonical reduced Hamiltonian, assume a canonical symplectic form $dq^i \wedge dp_i$ and compute the Hamilton's equations accordingly. Then compute Hamilton's equations for the noncanonical form with Hamiltonian (12) and transform the first set of equations via the transformation from p's to $\tilde{p}'$s. If the equations are the same, then our assumption of the canonical symplectic form for Eq. 7 was valid and this Hamiltonian is canonical:

$$\dot{R} = \frac{p_R}{m} \qquad \dot{p}_R = -\frac{\partial V}{\partial R} + \frac{(\gamma - p_a - p_b)^2}{mR^3} \tag{13}$$

$$\dot{\chi}_a = \frac{p_a}{I_a} - \frac{\gamma - p_a - p_b}{mR^2} \qquad \dot{p}_a = -\frac{\partial V}{\partial \chi_a} \tag{14}$$

$$\dot{\chi}_b = \frac{p_b}{I_b} - \frac{\gamma - p_a - p_b}{mR^2} \qquad \dot{p}_b = -\frac{\partial V}{\partial \chi_b} \tag{15}$$

The noncanonical symplectic form may be written locally as

$$\omega_\mu = \begin{pmatrix} -\mu B_{\alpha\beta} & -I \\ I & 0 \end{pmatrix} \tag{16}$$

From this we obtain the Hamilton's equations for the Hamiltonian in Eq. 12:

$$\nabla H = \omega_\mu \dot{z} \implies \begin{pmatrix} -\frac{(\tilde{p}_a+\tilde{p}_b)^2}{mR^3} - \frac{mR\mu^2}{I_{zz}^2} + \frac{\partial V}{\partial R} \\ \frac{\partial V}{\partial \chi_a} \\ \frac{\partial V}{\partial \chi_b} \\ \frac{\tilde{p}_R}{m} \\ \frac{\tilde{p}_a}{I_a} + \frac{\tilde{p}_a+\tilde{p}_b}{mR^2} \\ \frac{\tilde{p}_b}{I_b} + \frac{\tilde{p}_a+\tilde{p}_b}{mR^2} \end{pmatrix} = \begin{pmatrix} 0 & -B_a & -B_b & -1 & 0 & 0 \\ B_a & 0 & 0 & 0 & -1 & 0 \\ B_b & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{R} \\ \dot{\chi}_a \\ \dot{\chi}_b \\ \dot{\tilde{p}}_R \\ \dot{\tilde{p}}_a \\ \dot{\tilde{p}}_b \end{pmatrix}$$

Applying the coordinate transformation described in (13)-(15) these equations may be written as follows

$$\dot{\tilde{p}}_R = \dot{p}_R = -\dot{\chi}_a B_a - \dot{\chi}_b B_b + \frac{\mu^2 mR}{I_{zz}^2} + \frac{(\tilde{p}_a + \tilde{p}_b)^2}{mR^3} - \frac{\partial V}{\partial R} \qquad \dot{R} = \frac{\tilde{p}_R}{m} = \frac{p_R}{m} \tag{17}$$

$$\dot{\tilde{p}}_a = \dot{p}_a + \frac{2\mu m I_a R\dot{R}}{I_{zz}^2} = -\frac{\partial V}{\partial \chi_a} + \frac{2\mu m I_a R\dot{R}}{I_{zz}^2} \quad \dot{\chi}_a = \frac{\tilde{p}_a}{I_a} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} \tag{18}$$

$$\implies \dot{p}_a = -\frac{\partial V}{\partial \chi_a} \qquad = \frac{p_a}{I_a} - \frac{\mu}{I_{zz}} + \frac{p_a + p_b}{mR^2} - \frac{\mu I_a + \mu I_b}{I_{zz} mR^2} \tag{19}$$

$$= \frac{p_a}{I_a} - \frac{p_a + p_b}{mR^2} - \mu\frac{mR^2 + I_a + I_b}{mR^2 I_{zz}} \tag{20}$$

$$= \frac{p_a}{I_a} + \frac{p_a + p_b - \mu}{mR^2} \tag{21}$$

$$\dot{\tilde{p}}_b = \dot{p}_b + \frac{2\mu m I_b R\dot{R}}{I_{zz}^2} = -\frac{\partial V}{\partial \chi_b} + \frac{2\mu m I_b R\dot{R}}{I_{zz}^2} \quad \dot{\chi}_b = \frac{\tilde{p}_b}{I_b} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} \tag{22}$$

$$\implies \dot{p}_b = -\frac{\partial V}{\partial \chi_b} \qquad = \frac{p_b}{I_b} - \frac{\mu}{I_{zz}} + \frac{p_a + p_b}{mR^2} - \frac{\mu I_a + \mu I_b}{I_{zz} mR^2} \tag{23}$$

$$= \frac{p_b}{I_b} - \frac{p_a + p_b}{mR^2} - \mu\frac{mR^2 + I_a + I_b}{mR^2 I_{zz}} \tag{24}$$

$$= \frac{p_b}{I_b} + \frac{p_a + p_b - \mu}{mR^2} \tag{25}$$

Substitute into $\dot{\tilde{p}}_R$ the $\dot{\chi}_a$ and $\dot{\chi}_b$ values obtained from the reduced Hamilton's equations:

$$\dot{\tilde{p}}_R = -\dot{\chi}_a B_a - \dot{\chi}_b B_b + \frac{mR\mu^2}{I_{zz}^2} + \frac{(\tilde{p}_a + \tilde{p}_b)^2}{mR^2} - \frac{\partial V}{\partial R}$$

$$\begin{aligned}
\tilde{p}_a &= p_a - \frac{\mu I_a}{I_{zz}}, & \tilde{p}_b &= p_b - \frac{\mu I_b}{I_{zz}} \\
\dot{\chi}_a &= \frac{\tilde{p}_a}{I_a} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2}, & \dot{\chi}_b &= \frac{\tilde{p}_b}{I_b} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} \\
&= \frac{p_a}{I_a} + \frac{p_a + p_b - \mu}{mR^2} & &= \frac{p_b}{I_b} + \frac{p_a + p_b - \mu}{mR^2} \\
B_a &= \frac{2\mu m R I_a}{I_{zz}^2}, & B_b &= \frac{2\mu m R I_a}{I_{zz}^2}
\end{aligned}$$

$$\begin{aligned}
-\dot{\chi}_a B_a - \dot{\chi}_b B_b + \frac{mR\mu^2}{I_{zz}^2} &= -\frac{mR^2\mu^2 - 2mR^2\mu^2 - 2\mu^2(I_a + I_b)}{RI_{zz}^2} - \frac{2\mu(p_a + p_b)}{RI_{zz}} \\
&= -\frac{mR\mu^2}{I_{zz}^2} + \frac{2\mu^2}{RI_{zz}} - \frac{2\mu(p_a + p_b)}{RI_{zz}}
\end{aligned}$$

$$\begin{aligned}
\frac{(\tilde{p}_a + \tilde{p}_b)^2}{mR^3} &= \frac{\left((p_a + p_b) - \frac{\mu(I_a + I_b)}{I_{zz}}\right)^2}{mR^3} \\
&= \frac{(p_a + p_b)^2}{mR^3} - \frac{2\mu(I_a + I_b)(p_a + p_b)}{mR^3 I_{zz}} + \frac{\mu^2(I_a + I_b)^2}{mR^3 I_{zz}^2}
\end{aligned}$$

$$\dot{\tilde{p}}_R = -\frac{mR\mu^2}{I_{zz}^2} + \frac{2\mu^2}{RI_{zz}} - \frac{2\mu(p_a+p_b)}{RI_{zz}} + \frac{(p_a+p_b)^2}{mR^3} - \frac{2\mu(I_a+I_b)(p_a+p_b)}{mR^3 I_{zz}} + \frac{\mu(I_a+I_b)^2}{mR^3 I_{zz}^2} - \frac{\partial V}{\partial R} \tag{26}$$

$$= \frac{(p_a+p_b)^2}{mR^3} - \frac{2\mu(p_a+p_b)}{mR^3} + \frac{\mu^2}{mR^3 I_{zz}^2}\left(-(mR^2)^2 + 2mR^2 I_{zz} + (I_a+I_b)^2\right) - \frac{\partial V}{\partial R} \tag{27}$$

$$= \frac{(p_a+p_b)^2}{mR^3} - \frac{2\mu(p_a+p_b)}{mR^3} + \frac{\mu^2}{mR^3 I_{zz}^2}\left(mR^2 + I_a + I_b\right)^2 - \frac{\partial V}{\partial R} \tag{28}$$

$$= \frac{(\mu - p_a - p_b)^2}{mR^3} - \frac{\partial V}{\partial R} \equiv \dot{p}_R \tag{29}$$

Hence the Hamiltonian of Eq. 7 is canonical with canonical symplectic form. We also want to confirm that the Hamiltonian in body frame coordinates is canonical. As before, we assume that the symplectic form is canonical and compare with Eqs. 13-15.

$$\dot{R} = \frac{p_R}{m} \qquad \dot{p}_R = -\frac{\partial V}{\partial R} + \frac{(\gamma - p_\alpha)^2}{mR^3} \tag{30}$$

$$\dot{\alpha} = -\frac{\gamma - p_\alpha}{mR^2} + \frac{p_\alpha - p_\beta}{I_a} \qquad \dot{p}_\alpha = -\frac{\partial V}{\partial \alpha} \tag{31}$$

$$\dot{\beta} = -\frac{p_\alpha - p_\beta}{I_a} + \frac{p_\beta}{I_b} \qquad \dot{p}_\beta = -\frac{\partial V}{\partial \beta} \tag{32}$$

Adding the expressions for $\dot{\alpha}$ and $\dot{\beta}$ we recover the equation for $\dot{\chi}_b$. Since we obtain the cartesian coordinate (SE(2)) Hamiltonian from a polar transformation, it follows that this Hamiltonian is also canonical.

C7 Relative Equilibrium Conditions

First, we will compute the R.E. conditions for the canonical Hamiltonian of Eq. 7. We must note that by conservation of total angular momentum, $\dot{\theta} = \Omega$ is a constant. Thus the R.E. conditions become

$$\dot{R} = \frac{p_R}{m} = 0 \qquad \Longrightarrow \qquad p_R = 0 \tag{33}$$

$$\dot{p}_R = -\frac{\partial V}{\partial R} + \frac{(\gamma - p_a - p_b)^2}{mR^3} = 0 \qquad \Longrightarrow \qquad \frac{\partial V}{\partial R} = \frac{(\gamma - p_a - p_b)^2}{mR^3} = mR^2\Omega^2 \tag{34}$$

$$\dot{p}_a = -\frac{\partial V}{\partial \chi_a} = 0 \qquad \Longrightarrow \qquad \frac{\partial V}{\partial \chi_a} = 0 \tag{35}$$

$$\dot{p}_b = -\frac{\partial V}{\partial \chi_b} = 0 \qquad \Longrightarrow \qquad \frac{\partial V}{\partial \chi_b} = 0 \tag{36}$$

$$\dot{\chi}_a = \frac{p_a}{I_a} - \frac{\gamma - p_a - p_b}{mR^2} \qquad \Longrightarrow \qquad \dot{\chi}_a = 0, \dot{\theta} = \Omega \tag{37}$$

$$\dot{\chi}_b = \frac{p_b}{I_b} - \frac{\gamma - p_a - p_b}{mR^2} \qquad \Longrightarrow \qquad \dot{\chi}_b = 0, \dot{\theta} = \Omega \tag{38}$$

We must now compute the relative equilibrium conditions for the momentum shifted Hamiltonian and reconstruct these conditions from them.

$$\dot{R} = \frac{\tilde{p}_R}{m} = 0 \qquad \Longrightarrow \qquad \tilde{p}_R = 0 \tag{39}$$

$$\dot{\tilde{p}}_R = -\dot{\chi}_a B_a - \dot{\chi}_b B_b + \frac{\mu^2 mR}{I_{zz}^2} + \frac{(\tilde{p}_a + \tilde{p}_b)^2}{mR^3} - \frac{\partial V}{\partial R} \tag{40}$$

$$= \frac{\mu^2 mR}{I_{zz}^2} + \frac{(\tilde{p}_a + \tilde{p}_b)^2}{mR^3} - \frac{\partial V}{\partial R} \qquad \Longrightarrow \qquad \frac{\partial V}{\partial R} = \frac{\mu^2 mR}{I_{zz}^2} + \frac{(\tilde{p}_a + \tilde{p}_b)^2}{mR^3} \tag{41}$$

$$\dot{\tilde{p}}_a = -\frac{\partial V}{\partial \chi_a} + \frac{2\mu m I_a R\dot{R}}{I_{zz}^2} \qquad \Longrightarrow \qquad \frac{\partial V}{\partial \chi_a} = \frac{2\mu m I_a R\dot{R}}{I_{zz}^2} = 0 \tag{42}$$

$$\dot{\tilde{p}}_b = -\frac{\partial V}{\partial \chi_b} + \frac{2\mu m I_b R\dot{R}}{I_{zz}^2} \qquad \Longrightarrow \qquad \frac{\partial V}{\partial \chi_b} = \frac{2\mu m I_b R\dot{R}}{I_{zz}^2} = 0 \tag{43}$$

$$\dot{\chi}_a = \frac{\tilde{p}_a}{I_a} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} \qquad \Longrightarrow \qquad \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} = -\frac{\tilde{p}_a}{I_a} \tag{44}$$

$$\dot{\chi}_b = \frac{\tilde{p}_b}{I_b} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} \qquad \Longrightarrow \qquad \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} = -\frac{\tilde{p}_b}{I_b} \tag{45}$$

From these conditions we immediately recover the conditions

$$\tilde{p}_R = p_R = 0, \qquad \frac{\partial V}{\partial \chi_a} = 0, \qquad \frac{\partial V}{\partial \chi_b} = 0$$

and we have $\frac{\tilde{p}_a}{I_a} = \frac{\tilde{p}_b}{I_b}$ which implies $\tilde{p}_a = \tilde{p}_b = 0$ (see below). The condition on $\dot{\tilde{p}}_R$ becomes

$$\begin{aligned}\frac{\partial V}{\partial R} &= \frac{\mu^2 mR}{I_{zz}^2} + \frac{mR^2\tilde{p}_a^2}{I_a^2} \\ &= \frac{\mu^2 mR}{I_{zz}^2}\end{aligned}$$

In other words, $\frac{\partial V_\mu}{\partial R} = 0$. This is summed up as

$$\tilde{p}_R = 0, \qquad \tilde{p}_a = 0, \qquad \tilde{p}_b = 0 \tag{46}$$

$$\frac{\partial V_\mu}{\partial R} = 0, \qquad \frac{\partial V_\mu}{\partial \chi_a} = 0, \qquad \frac{\partial V_\mu}{\partial \chi_b} = 0. \tag{47}$$

To see that $\tilde{p}_a = \tilde{p}_b = 0$, let $\tilde{p}_b = \frac{I_b}{I_a}\tilde{p}_a$ and substitute into the expression for $\dot{\chi}_a$:

$$\begin{aligned}\frac{\tilde{p}_a}{I_a} + \frac{\tilde{p}_a + \tilde{p}_b}{mR^2} &= \tilde{p}_a\left(\frac{1}{I_a} + \frac{1}{mR^2}\right) + \tilde{p}_a\frac{I_b}{I_a mR^2} \\ &= \tilde{p}_a\left(\frac{mR^2 + I_a + I_b}{I_a mR^2}\right) = \frac{I_{zz}\tilde{p}_a}{I_a mR^2} = 0\end{aligned}$$

Since I_{zz} is non-zero, it follows that $\tilde{p}_a = 0$. The expressions $\frac{\partial V_\mu}{\partial \chi_a} = \frac{\partial V_\mu}{\partial \chi_b} = 0$ and $\tilde{p}_R = 0$ reconstruct exactly the expressions $\frac{\partial V}{\partial \chi_a} = \frac{\partial V}{\partial \chi_b} = p_R = 0$. Attempting to reconstruct the remaining conditions, we come across the following equalities:

$$\frac{\gamma - p_a - p_b}{mR^2} = \Omega = \frac{\mu}{I_{zz}}$$

It is clear that $\Omega = \frac{\gamma - p_a - p_b}{mR^2}$, so only the second equality must be checked.

C8 Abelian Reduction - Lagrangian Side

- **Infinitesimal generator:** The infinitesimal generator corresponding to the cyclic action of $\theta \in S^1$ is

$$\xi_Q(\theta, q^i) = \left.\frac{d}{dt}\right|_{t=0} \left(e^{t\xi} \cdot (\theta, q^\alpha)\right) = ((\theta, q^\alpha), (\xi, 0))$$

- **Lagrangian momentum map:** The momentum map associated to this infinitesimal generator is $\mathbf{J}_L : TQ \to \mathfrak{g}^*$

$$\begin{aligned}\mathbf{J}_L((q,\dot{q})) \cdot \xi_Q = \langle \mathbb{F}L(q,\dot{q}), \xi_Q(q)\rangle = \langle (g_{0j}\dot{q}^j, g_{ij}\dot{q}^j), (\xi, 0)\rangle &= g_{0j}\dot{q}^j\xi \\ &= g_{00}\dot{\theta} + g_{0\alpha}\dot{q}^\alpha \\ &= \left(mR^2 + I_a + I_b\right)\dot{\theta} + I_a\dot{\chi}_a + I_b\dot{\chi}_b\end{aligned}$$

- **Locked inertia tensor:** This is the instantaneous tensor of inertia calculated when the relative motion of the bodies is fixed. We introduce $\langle\langle \cdot, \cdot \rangle\rangle$ as the inner product induced from g_{ij}. Using this we may write $\mathbb{I}(\theta, q^\alpha) : \mathfrak{g} \to \mathfrak{g}^*$ (locally) as:

$$\langle \mathbb{I}(\theta, q^\alpha)\eta, \xi\rangle - \langle\langle ((\theta, q^\alpha), (\eta, 0)), ((\theta, q^\alpha), (\xi, 0))\rangle\rangle = g_{00}\eta\xi.$$

  so that $\mathbb{I}(\theta, q^\alpha) = g_{00}(q^\alpha) = mR^2 + I_a + I_b$.

- **Mechanical connection:** We may write the connection $\mathcal{A} : TQ \to \mathfrak{g}$ (locally) as $\mathcal{A}(\theta, q^\alpha)(\theta, q^\alpha, \dot{\theta}, \dot{q}^\alpha) = \mathbb{I}^{-1}\mathbf{J}(\mathbb{F}L(\theta, q^\alpha, \dot{\theta}, \dot{q}^\alpha)) = g_{00}^{-1}g_{0j}\dot{q}^j$. This simplifies as

$$\begin{aligned}\mathcal{A}(\theta, q^\alpha)(\theta, q^\alpha, \dot{\theta}, \dot{q}^\alpha) &= \dot{\theta} + g_{00}^{-1}g_{0\alpha}\dot{q}^\alpha \\ &= \dot{\theta} + \frac{I_a\dot{\chi}_a + I_b\dot{\chi}_b}{mR^2 + I_a + I_b}\end{aligned}$$

  From $\mathcal{A}$ we can obtain the one form: $\mathcal{A}(\theta, q^\alpha) = d\theta + A_\alpha dq^\alpha$, where $A_\alpha = g_{00}^{-1}g_{0\alpha}$:

$$\mathcal{A}(\theta, q^\alpha) = d\theta + \frac{I_a}{mR^2 + I_a + I_b}d\chi_a + \frac{I_b}{mR^2 + I_a + I_b}d\chi_b$$

  The curvature $\mathcal{B} = d\mathcal{A} = B_{\alpha\beta}dq^\alpha \wedge dq^\beta$ has local components $B_{\alpha\beta} = \left(\frac{\partial A_\alpha}{\partial q^\beta} - \frac{\partial A_\beta}{\partial q^\alpha}\right)$ which may be written in (R, χ_a, χ_b) coordinates as the following matrix:

$$B_{\alpha\beta} = \begin{pmatrix} 0 & -\frac{\partial A_a}{\partial R} & -\frac{\partial A_b}{\partial R} \\ \frac{\partial A_a}{\partial R} & 0 & \left(\frac{\partial A_a}{\partial \chi_b} - \frac{\partial A_b}{\partial \chi_a}\right) \\ \frac{\partial A_b}{\partial R} & -\left(\frac{\partial A_a}{\partial \chi_b} - \frac{\partial A_b}{\partial \chi_a}\right) & 0 \end{pmatrix} = \begin{pmatrix} 0 & \frac{2mRI_a}{(mR^2+I_a+I_b)^2} & \frac{2mRI_b}{(mR^2+I_a+I_b)^2} \\ -\frac{2mRI_a}{(mR^2+I_a+I_b)^2} & 0 & 0 \\ -\frac{2mRI_b}{(mR^2+I_a+I_b)^2} & 0 & 0 \end{pmatrix}$$

  For a given $\mu \in \mathfrak{g}^* \cong \mathbb{R}$, the mechanical connection on the fiber $Q \to Q/G$ is

$$\begin{aligned}\alpha_\mu(\theta, q^\alpha) &= \mu d\theta + \mu A_\alpha dq^\alpha \\ &= \mu d\theta + \frac{\mu I_a}{mR^2 + I_a + I_b}d\chi_a + \frac{\mu I_b}{mR^2 + I_a + I_b}d\chi_b\end{aligned}$$

- **Amended potential:** For $\mu \in \mathfrak{g}^*$, the amended potential is defined as

$$V_\mu(q) = V(q) + \frac{1}{2}\langle \mu, \mathbb{I}^{-1}(q)\mu\rangle = V(q) + \frac{1}{2} g_{00}^{-1}\mu^2$$
$$= V(q) + \frac{\mu^2}{2\left(mR^2 + I_a + I_b\right)}$$

- **Routhian:** The Routhian is a function on TQ defined as

$$R^\mu = \frac{1}{2}\|\text{Hor}(q,\dot{q})\|^2 - V_\mu$$

where $\text{Hor}(q,\dot{q}) = (-g_{00}^{-1}g_{0\alpha}\dot{q}^\alpha, \dot{q}^\alpha)$ is the horizontal component of $(q,\dot{q})$ and the norm is induced from g_{ij}. We may write this locally as:

$$R^\mu = \frac{1}{2}\left(g_{\alpha\beta} - g_{00}^{-1}g_{0\alpha}g_{0\beta}\right)\dot{q}^\alpha\dot{q}^\beta - \frac{1}{2}g_{00}^{-1}\mu^2 - V(q^\alpha)$$
$$= \frac{1}{2}h_{\alpha\beta}\dot{q}^\alpha\dot{q}^\beta - \frac{\mu^2}{2\left(mR^2 + I_a + I_b\right)} - V(q^\alpha)$$

where $h_{\alpha\beta}$ is as follows

$$h_{\alpha\beta} = g_{\alpha\beta} - g_{00}^{-1}g_{0\alpha}g_{0\beta} = \begin{pmatrix} m & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_b \end{pmatrix} - \frac{1}{mR^2 + I_a + I_b}\begin{pmatrix} 0 & 0 & 0 \\ 0 & I_a^2 & I_aI_b \\ 0 & I_aI_b & I_b^2 \end{pmatrix}$$
$$= \begin{pmatrix} m & 0 & 0 \\ 0 & \frac{I_amR^2 + I_aI_b}{mR^2+I_a+I_b} & -\frac{I_aI_b}{mR^2+I_a+I_b} \\ 0 & -\frac{I_aI_b}{mR^2+I_a+I_b} & \frac{I_bmR^2+I_aI_b}{mR^2+I_a+I_b} \end{pmatrix}$$

Working these calculations out explicitly we find the following Routhian:

$$\hat{R} = \frac{\dot{\chi}_a^2(I_aI_b + I_amR^2)}{2\mathbb{I}} + \frac{\dot{\chi}_b^2(I_aI_b + I_bmR^2)}{2\mathbb{I}} + \frac{m\dot{R}^2}{2} - \frac{\dot{\chi}_a\dot{\chi}_bI_aI_b}{\mathbb{I}} - V_\mu(R,\chi_a,\chi_b) \tag{48}$$

from which we obtain the Lagrange-Routh equations:

$$m\ddot{R} - \frac{(\dot{\chi}_aI_a + \dot{\chi}_bI_b)^2mR}{\mathbb{I}^2} + \frac{2\mu mR(\dot{\chi}_aI_a + \dot{\chi}_bI_b)}{\mathbb{I}^2} = -\frac{\partial V_\mu}{\partial R} \tag{49}$$

$$\frac{(\ddot{\chi}_a - \ddot{\chi}_b)I_aI_b + \ddot{\chi}_aI_amR^2 + 2mRI_a\dot{\chi}_a\dot{R}}{\mathbb{I}} - \frac{2mR\dot{R}((\ddot{\chi}_a - \dot{\chi}_b)I_aI_b + \mu I_a + \dot{\chi}_aI_amR^2)}{\mathbb{I}^2} = -\frac{\partial V_\mu}{\partial \chi_a} \tag{50}$$

$$\frac{(\ddot{\chi}_b - \ddot{\chi}_a)I_aI_b + \ddot{\chi}_bI_bmR^2 + 2mRI_b\dot{\chi}_b\dot{R}}{\mathbb{I}} - \frac{2mR\dot{R}((\ddot{\chi}_b - \dot{\chi}_a)I_aI_b + \mu I_b + \dot{\chi}_bI_bmR^2)}{\mathbb{I}^2} = -\frac{\partial V_\mu}{\partial \chi_b} \tag{51}$$

C9 Abelian Reduction - Hamiltonian Side

We have a Hamiltonian which is written locally as

$$H(q^\alpha, p_\theta, p_q^\alpha) = \frac{1}{2}g^{00}p_\theta^2 + g^{0\alpha}p_\theta p_\alpha + \frac{1}{2}g^{\alpha\beta}p_\alpha p_\beta + V(q^\alpha)$$

- **Momentum map:** The momentum map corresponds to the angular momentum of the system: $\mathbf{J} : T^*Q \to \mathfrak{g}^*$

$$\langle \mathbf{J}(\theta, q^\alpha, p_\theta, p_r^\alpha), \xi \rangle = \langle (p_\theta, p_\alpha), (\xi, 0) \rangle = p_\theta \xi.$$

so that $\mathbf{J}(\theta, q^\alpha, p_\theta, p_r^\alpha) = p_\theta = l + p_a + p_b \equiv J$.

- **Momentum shifting:** We shift the momenta from $\mathbf{J}^{-1}(\mu)$ to $\mathbf{J}^{-1}(0)$, both in the full space and in reduced space. The momentum shifting is given by

$$\begin{aligned}
\tilde{p}_\theta &= 0 \\
\tilde{p}_R &= p_R \\
\tilde{p}_a &= p_a - \frac{\mu I_a}{mR^2 + I_a + I_b} \\
\tilde{p}_b &= p_b - \frac{\mu I_b}{mR^2 + I_a + I_b}
\end{aligned}$$

- **Reduced Hamiltonian:** In $\mathbf{J}^{-1}(0)/G$, we have $H_{\alpha_\mu} = \frac{1}{2}\|\tilde{p}\|^2 + V_\mu$ where the norm is induced from the metric and V_μ is the amended potential. Since $\tilde{p}_\theta = 0$ we have the following Hamiltonian in the reduced space $\mathbf{J}^{-1}(0)/G$:

$$H_\mu(q^\alpha, p_\alpha) = \frac{1}{2}g^{\alpha\beta}\tilde{p}_\alpha\tilde{p}_\beta + V(q^\alpha) + \frac{\mu^2}{2}g_{00}^{-1}$$

Written explicitly, the reduced Hamiltonian is as follows:

$$H_\mu = \frac{\tilde{p}_R^2}{2m} + \frac{\tilde{p}_a^2}{2I_a} + \frac{\tilde{p}_b^2}{2I_b} + \frac{(\tilde{p}_a + \tilde{p}_b)^2}{2mR^2} + V_\mu(R, \chi_a, \chi_b) \tag{52}$$

$$= \frac{(I_a + I_b)\mu^2}{(2mR^2)(I_a + I_b + mR^2)} - \frac{(p_a + p_b)\mu}{mR^2} + \frac{(p_a + p_b)^2}{2mR^2} + \frac{R^2(I_b m p_a^2 + I_a m p_b^2 + I_a I_b p_R^2)}{2mI_a I_b R^2} + V_\mu \tag{53}$$

- **Reduced symplectic form:** In general, in the reduced space, the symplectic form is not canonical. The projection is given by the map:

$$((T^*Q)_\mu, \Omega_\mu)\overrightarrow{P_\mu}((T^*(Q/G), \omega - B_\mu)$$

where the reduced symplectic form is

$$\begin{aligned}
\omega_\mu = \omega - B_\mu &= dq^\alpha \wedge d\tilde{p}_\alpha - \mu\frac{\partial A_\alpha}{\partial q^\beta}dq^\beta \wedge dq^\alpha \\
&= dR \wedge d\tilde{p}_R + d\chi_a \wedge d\tilde{p}_a + d\chi_b \wedge d\tilde{p}_b + \frac{2\mu m R}{(mR^2 + I_a + I_b)^2}\left(I_a dR \wedge d\chi_a + I_b dR \wedge d\chi_b\right)
\end{aligned}$$

Appendix D Lifetime Distribution Code

The code to compute lifetime distributions for the Rydberg scattering reaction and planar H_2O-H_2 scattering reaction is written in C++ and compiled using gcc version 3.4.5 on a PC running Redhat Linux. The software is broken into five major components, contained in five separate files: fullscatter.cc, newt.cc, rk78.cc, moduls.h, and makefile.

fullscatter.cc - This contains the main loop which computes lifetime distributions for varying system parameters, such as energy H, total angular momentum J, and electric field strength ϵ.

newt.cc - All supporting functions, except the integrator, are contained in this file. Supporting functions include the Hamiltonian and Hamiltonian vector fields, bounding box functions, sampling code and functions used to project sampled points onto the energy surface.

rk78.cc - Runga Kutta 78 integrator that I inherited from Frederic Gabern during his Postdoctoral work at Caltech.

moduls.h - This file contains static system parameters, such as π, moments of inertia of H_2O and H_2, among others.

makefile - The makefile tells the compiler how to combine code from the previous four files to create an executable binary.

The code for the dual method test is very similar to fullscatter.cc, except that the order of all integration is reversed, along with the necessary conditions for testing reactivity. The file is not included in this appendix, however the code may be obtained by contacting the author.

D1 fullscatter.cc - Main Scattering Code

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "newt.h"   //Methods, Hamiltonian, Vector Field, etc.
#include "rk78.h"   //Integrator Code
#include "moduls.h" //System Parameters

#if !defined(DATA)
#define DATA ""
#endif

/*-Lifetime Distribution Parameters-*/
#define BINS 30
#define CHUNKSIZE 1000
#define MAXCHUNKS 1000
#define SDTOL .001
/*-Epsilon Parameters-*/
#define epsMIN  .58
#define epsSTEP .005
#define STEPS 5

/*---ORGANIZATION OF CODE---see section 2 of thesis for details on method-----//

  * STEP 0 - INITIALIZE VARIABLES, INTEGRATOR, & MAIN LOOP 1

  * MAIN LOOP 1 (COMPUTES LDs FOR DIFFERENT VALUES OF EPSILON) {

        * SET EPSILON & FILE POINTERS

        * STEP I - IDENTIFY FIXED POINT AND CUTS

        * STEP IIa - DETERMINE BOUNDING BOX AT OUTCUT (NAMED "cut")

        * MAIN LOOP 2 (COMPUTES LD DATA UNTIL SD OF EACH BIN IS < SDTOL) {

                * MAIN LOOP 3 (COMPUTES LD DATA IN CHUNKS OF SIZE CHUNKSIZE) {

                        * STEP IIb - SELECT REACTIVE TRAJECTORIES

                        * STEP III - INTEGRATE INTO BOUND STATE UNTIL ESCAPE

                } CONTINUE ML3 UNTIL CHUNKSIZE TRAJECTORIES HAVE BEEN TESTED

                * STEP 4 - WRITE DATA CHUNK TO FILES

                * STEP 5a - COMPUTE SD OF EACH BIN CHUNKWISE

                * STEP 5b - FIND THE LARGEST SD OF ALL BINS (CALL IT SDMAX)

        } CONTINUE ML2 UNTIL SDMAX < SDTOL OR UNTIL #CHUNKS EXCEEDS MAXCHUNKS

        * STEP 6 - WRITE LIFETIME DISTRIBUTION FOR SPECIFIC EPSILON

  } CONTINUE ML1 UNTIL LDs ARE COMPUTED FOR ALL SELECTED VALUES OF EPSILON

  * STEP 7 - UNINITIALIZE INTEGRATOR
//-------------------------------------------------------------------------*/
```

```
int main(void)
{
/*
   STEP 0 - INITIALIZING VARIABLES, INTEGRATOR, & MAIN LOOP 1
*/
   int i,j,k,outs,rpoints,letters,epscount; //counters
   double epsilon;  //field strength or total angular momentum parameter.
   double t,h,hmin,hmax;  //integrator parameters
   double inincut,incut,fpcut,cut,outcut,cutdiff;  //cuts
   double min[5],max[5],omin[5],omax[5]; //bounding box dimensions
   double saddle[6],randx[6],randxs[6];  //phase space vectors
   double pic[CHUNKSIZE][6],opic[CHUNKSIZE][6];  //image of reactive trajectories
   double picsave[CHUNKSIZE][6],opicsave[CHUNKSIZE][6],timesave[CHUNKSIZE];  //" "
   int orbit,orbitsave[CHUNKSIZE];  //saves orbit data for entire chunk of data
   char strOCI [50],strIFO [50],strLD [50],strSC [50], strFP [50];  //strings holding
file names
   FILE *OCI,*IFO,*LD,*SC,*FP;  //file pointers
   /*-bin variables-*/
   int binLD[BINS];  //combined bin data (cummulative count)
   int bin[MAXCHUNKS][BINS];  //bin data for each data chunk
   int chunk;  //keeps track of which data chunk we are on
   int intraj;  //whether or not trajectory is reactive
   double maxsd;  //largest SD of all bins
   double meanLD[BINS];  //average of binLD over all points
   double stdevLD[BINS];  //SD of each bin chunkwise
   /*-random number generator-*/
   srand((unsigned) time(NULL) );
   testrand();
   /*-integrator-(rk78.cc)-*/
   ini_rk78(6);
/*
   MAIN LOOP 1 (COMPUTES LDs FOR DIFFERENT VALUES OF EPSILON)
*/
   for(epscount = 0;epscount < STEPS; epscount++) {
     epsilon = epsMIN + epscount*epsSTEP;
     letters = sprintf(strOCI,"./fullscatter/OCI/OUTScomingIN_%e.oci.001",epsilon);
     letters = sprintf(strIFO,"./fullscatter/IFO/INSfromOUT_%e.ifo.001",epsilon);
     letters = sprintf(strLD,"./fullscatter/LD/distribution_%e.ld.001",epsilon);
     letters = sprintf(strSC,"./fullscatter/scatter_%e.sc.001",epsilon);
     letters = sprintf(strFP,"./fullscatter/fixedpoint.dad");
   /*
      STEP 1 - IDENTIFY FIXED POINT AND CUTS
   */
     FP=fopen(strFP,"r");
     if(FP==NULL) {printf("fullscatter: Can't open %s\n",strFP);exit(1);}
     for(i=0;i<6;i++) fscanf(FP,"%le ",&saddle[i]);
     fclose(FP);
     cutdiff = .1;
     fpcut = saddle[0];
     if(saddle[0]<0) cutdiff = -1.*cutdiff;
     inincut = fpcut - 1.2*cutdiff;
     incut = fpcut - cutdiff;
     cut = fpcut + cutdiff;
     outcut = fpcut + 1.2*cutdiff;
/*-alternate code for H2O-H2 scattering-*/
/*
     R = Req();
```

```
     Om = Omeq(R);
     l = leq(R);
     gam = l+Om*(Ia+Ib);
     saddle[0] = R, saddle[1] = PI, saddle[2] = 0,saddle[3] = 0,saddle[4] =
Om*(Ia+Ib),saddle[5] = Om*Ib;
     epsilon = gam - epscount*epsSTEP;
     //CALCULATING CUTS
     cutdiff = 2.5;
     fpcut = saddle[0];
     if(saddle[0]<0) {
       inincut = fpcut + 1.2*cutdiff;
       incut = fpcut + cutdiff;
       cut = fpcut - cutdiff;
       outcut = fpcut - 2*cutdiff;
     }
     else {
       inincut = fpcut - 1.2*cutdiff;
       incut = fpcut - cutdiff;
       cut = fpcut + cutdiff;
       outcut = fpcut + 2*cutdiff;
     }
*/
   /*
      STEP 2a - DETERMINE BOUNDING BOX AT OUTCUT (NAMED "cut")
   */
     /*-Determining Rough Bounding Box @ fpcut-*/
     roughbox(saddle,min,max,1.e-5,1.e+0,24,20,5,epsilon);
     /*-Refining Rough Box @ fpcut-*/
     refinebox(min,max,incut,fpcut,cut,epsilon);
     /*-Computing Rough Bounding Box @ cut by Integrating Outs Backwards-*/
     newcutbox(omin,omax,min,max,fpcut,cut,incut,500,epsilon);
     /*-Refining Rough Box @ cut-*/
     refinebox(omin,omax,incut,cut,outcut,epsilon);
     /*-Cleaning Bin Variables Before New Run-*/
     maxsd = 1.;
     chunk = 0;
     for(i=0;i<BINS;i++) {
        binLD[i] = 0;
        for(j=0;j<MAXCHUNKS;j++) {
           bin[j][i] = 0;
        }
        meanLD[i] = 0.;
        stdevLD[i] = 1.;
     }
   /*
      MAIN LOOP 2 (COMPUTES LD DATA UNTIL SD OF EACH BIN IS < SDTOL)
   */
     while((chunk<MAXCHUNKS)&&(maxsd>SDTOL)) {
          i=0;
          outs = 0;
      /*
         MAIN LOOP 3 (COMPUTES LD DATA IN CHUNKS OF SIZE CHUNKSIZE)
      */
          while(i<CHUNKSIZE) {
               intraj = 0;
               t=0.e0;
               h=1.e-2;
               hmin=1.e-3;
```

```
                hmax = 1.e+1;
                rpoints = 0;
          /*
             STEP 2b - SELECT REACTIVE TRAJECTORIES
          */
                /*-Select Point @ Random from BB-*/
                while(rpoints==0) {
                     randompoint(randx,omin,omax,cut);
                     rpoints = energyfit3(randx,1.e-15,omin,omax,epsilon);
                }
                /*-Test Point for Reactivity-*/
                for(j=0;j<6;j++) randxs[j] = randx[j];
                while((randx[0]<outcut)&&(randx[0]>incut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,ffield,epsilon);
                if(randx[0]>=outcut) outs++;
                if(randx[0]<=incut) {
                  for(k=0;k<6;k++) {
                     pic[i][k] = randxs[k];
                     opic[i][k] = randx[k];
                     picsave[i][k] = randxs[k];
                     opicsave[i][k] = randx[k];
                  }
                  intraj = 1;
                }
                /*-If Point is Reactive, Run Through Bound State-*/
                if((intraj==1)&&(fabs(ham(pic[i],epsilon)+1.52)<.0001)) {
          /*
             STEP 3 - INTEGRATE INTO BOUND STATE UNTIL ESCAPE
          */
                  orbit = 0;
                  t=0.e0;
                  h=1.e-2;
                  hmin=1.e-4;
                  hmax=1.e+1;
                  /*-Integrate Point until it Leaves Bound State-*/
                  while((orbit<BINS)&&(fabs(pic[i][0])<2)) {
                       orbit += rk78(&t,pic[i],&h,1.e-15,hmin,hmax,6,ffield,epsilon);
                  }
                  /*-Record Number of Orbits-*/
                  orbitsave[i] = orbit;
                  timesave[i] = t;
                  if(orbit<BINS) {
                    binLD[orbit]++;
                    bin[chunk][orbit]++;
                  }
                  else {
                    binLD[BINS-1]++;
                    bin[chunk][BINS-1]++;
                  }
                  i++;
                }
           }//MAIN LOOP 3 - CONTINUE UNTIL CHUNKSIZE TRAJECTORIES HAVE BEEN TESTED
        /*
           STEP 4 - WRITE DATA CHUNK TO FILES
        */
           OCI = fopen(strOCI,"a");
           if(OCI == NULL) {printf("cannot open %s\n",strOCI); exit(1);}
           for(i=0;i<CHUNKSIZE;i++) {
```

```
            fprintf(OCI,"%d ",chunk*CHUNKSIZE+i);
            for(j=0;j<6;j++) fprintf(OCI,"%24.16e ",picsave[i][j]);
            fprintf(OCI,"%24.16e\n",ham(picsave[i],epsilon));
         }
         fclose(OCI);
         IFO = fopen(strIFO,"a");
         if(IFO == NULL) {printf("cannot open %s\n",strIFO); exit(1);}
         for(i=0;i<CHUNKSIZE;i++) {
            fprintf(IFO,"%d ",chunk*CHUNKSIZE+i);
            for(j=0;j<6;j++) fprintf(IFO,"%24.16e ",opicsave[i][j]);
            fprintf(IFO,"%24.16e\n",ham(opicsave[i],epsilon));
         }
         fclose(IFO);
         SC = fopen(strSC,"a");
         if(SC == NULL) {printf("cannot open %s\n",strSC); exit(1);}
         for(i=0;i<CHUNKSIZE;i++) {
            fprintf(SC,"%d %24.16e ",chunk*CHUNKSIZE+i,ham(pic[i],epsilon));
            fprintf(SC,"%24.16e %d\n",timesave[i],orbitsave[i]);
         }
         fclose(SC);
      /*
         STEP 5a - COMPUTE SD OF EACH BIN CHUNKWISE
      */
         for(i=0;i<BINS;i++) {
            meanLD[i] = 1.*binLD[i]/(CHUNKSIZE*(chunk+1));
            stdevLD[i] = 0;
            for(j=0;j<chunk;j++) {
               stdevLD[i] +=
(meanLD[i]-1.*bin[j][i]/CHUNKSIZE)*(meanLD[i]-1.*bin[j][i]/CHUNKSIZE);
            }
            stdevLD[i] = 1.*sqrt(stdevLD[i])/(chunk+1.);
         }
         if(chunk>0) maxsd = 0;
      /*
         STEP 5b - FIND THE LARGEST SD OF ALL BINS (CALL IT SDMAX)
      */
         for(i=0;i<BINS;i++) {
            if(stdevLD[i]>maxsd) maxsd = stdevLD[i];
         }
         printf("chunk = %d, sd = %e\n",chunk,maxsd);
         chunk++;
    }//MAIN LOOP 2 - CONTINUE UNTIL SDMAX < SDTOL OR UNTIL #CHUNKS EXCEEDS MAXCHUNKS
  /*
     STEP 6 - WRITE LIFETIME DISTRIBUTION FOR SPECIFIC EPSILON
  */
    LD = fopen(strLD,"w");
    for(i=0;i<12;i++) printf("%le\n",meanLD[i]);
    if(LD == NULL) {printf("cannot open %s\n",strLD); exit(1);}
    for(i=0;i<BINS;i++) fprintf(LD,"%d %24.16e %24.16e\n",i,meanLD[i],stdevLD[i]);
  }//MAIN LOOP 1 - CONTINUE UNTIL LDs ARE COMPUTED FOR ALL SELECTED VALUES OF EPSILON
/*
  STEP 7 - UNINITIALIZE INTEGRATOR
*/
  end_rk78(6);
  return(0);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "rk78.h"
#include "moduls.h"

/*-----

-NOTES: 1. If using H2O-H2 model, change ffieldH2OH2->ffield, bfieldH2OH2->bfield
             and hamH2OH2->ham.  Also, roughbox will need to be modified to use pRmax,
             pamin, pamax, pbmin, and pbmax

        2. Any function whose name ends in B is the backward integration version of
             another function, and is used for the dual method test

        3. Code is arranged in 4 major blocks:

            1 - RYDBERG SPECIFIC FUNCTIONS
            2 - H2O-H2 SPECIFIC FUNCTIONS
            3 - TEST/DIAGNOSTIC FUNCTIONS
            4 - FUNCTIONS THAT REMAIN UNCHANGED WITH NEW MODELS

-----*/

/*-------------------------------------*/
// RYDBERG SPECIFIC FUNCTIONS:
//   FUNCTIONS:
//   V(x,y,z,eps) - Potential Energy
//   ham(x[6],eps) - Hamiltonian
//   ffield(t,x[6],n,xdot[6],eps) - Forward Hamiltonian Vector Field
//   bfield(t,x[6],n,xdot[6],eps) - Backward Hamiltonian Vector Field
//   VARIABLES:
//   x[6] - phase space vector
//   xdot[6] - value of vector field at x[6]
//   eps - system parameter (angular momentum, field strength, etc.)
//   t,n - (unused) variables for time (t) or condition (n) dependant vector fields
/*-------------------------------------*/

double V(double x,double y,double z,double epsilon)
{
  /*
      V: Evaluates the Potential Energy at (x,y,z) with system parameter epsilon.
  */
  double r,v_eps;
  r = sqrt(x*x+y*y+z*z);
  v_eps = -1./r-epsilon*x;
  return(v_eps);
}

double ham(double x[6],double epsilon)
{
  /*
      ham: Evaluates the Hamiltonian at point x[6] with system parameter epsilon.
  */
  double V(double x, double y, double z,double epsilon);
  double H;
  H = (x[3]*x[3]+x[4]*x[4]+x[5]*x[5])/2 + (x[0]*x[4]-x[1]*x[3])/2 +
```

```
(x[0]*x[0]+x[1]*x[1])/8;
  H += V(x[0],x[1],x[2],epsilon);
  return(H);
}

void ffield(double t,double *x,int n,double *f,double epsilon)
{
  /*
      ffield: Evaluates the Ham. Vector Field at point x[6],
                parameter epsilon, time t, & condition n
  */
  double r,r3;
  r = sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);
  r3 = pow(r,3);

  f[0] = x[3]-x[1]/2;
  f[1] = x[4]+x[0]/2;
  f[2] = x[5];
  f[3] = -x[4]/2 - x[0]/4 - x[0]/r3 + epsilon;
  f[4] = x[3]/2 - x[1]/4 - x[1]/r3;
  f[5] = -x[2]/r3;
}

void bfield(double t, double *x,int n,double *f,double epsilon)
{
  /*
     bfield: Evaluates the Neg. Ham. Vector Field at point x[6], etc.
  */
  double r,r3;
  r = sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);
  r3 = pow(r,3);

  f[0] = -x[3]+x[1]/2;
  f[1] = -x[4]-x[0]/2;
  f[2] = -x[5];
  f[3] = x[4]/2 + x[0]/4 + x[0]/r3 - epsilon;
  f[4] = -x[3]/2 + x[1]/4 + x[1]/r3;
  f[5] = x[2]/r3;
}

/*-------------------------------------*/
// H2O-H2 SPECIFIC FUNCTIONS:
//   FUNCTIONS:
//   Req() - Equilibrium radius (of saddle)
//   Omeq(req) - Equilibrium Omega (at saddle)
//   leq(req) - Equilibrium l (at saddle)
//   h2oh2(r,a,b) - Potential Energy
//   hamH2OH2(x[6],eps) - Hamiltonian
//   ffieldH2OH2(t,x[6],n,xdot[6],eps) - Forward Hamiltonian Vector Field
//   bfieldH2OH2(t,x[6],n,xdot[6],eps) - Backward Hamiltonian Vector Field
//   pRmax(x[6],eps) - Computes max pR can drift from fp and hit energy surface
//   pamin(x[6],eps) - "" min pa ""
//   pamax(x[6],eps) - "" max pa ""
//   pbmin(x[6],eps) - "" min pb ""
//   pbmax(x[6],eps) - "" max pb ""
//   VARIABLES:
//   req - equilibrium radius
//   x[6] - phase space vector
```

```
//   xdot[6] - value of vector field at x[6]
//   eps - system parameter (angular momentum, field strength, etc.)
//   t,n - (unused) variables for time (t) or condition (n) dependant vector fields
/*-------------------------------------*/

double Req()
{
  /*
    Req: Solves for Req using the bi-quartic expression of Wiesenfeld.
         The quartic is in depressed form:
         R^8+(Cl/E0)R^4+(2/E0)*((Ia+Ib)Cl/m-Dl)R^2-3(Ia+Ib)Dl/mE0=0.
         The solution comes from http://en.wikipedia.org/wiki/Quartic_equation
  */
  double r2,c0,c1,c2;
  double p,q,r,u,y;
  double t1,t2;

  t1 = (Ia+Ib);
  t1 = t1/mass;
  t2 = 1/E0;

  c0 = -3.*Dl*t1*t2;
  c1 = (2.*Cl*t1-2.*Dl)*t2;
  c2 = Cl/E0;

  p = -c2*c2/12-c0;
  q = -c2*c2*c2/108+c2*c0/3-c1*c1/8;
  r = q/2+sqrt(q*q/4+p*p*p/27);
  if(r<0) u = -1.*pow(-1.*r,1./3);
  else u = pow(r,1./3);
  y = -5*c2/6-u+p/(3*u);
  r2 = (sqrt(c2+2*y)+sqrt(-(3*c2+2*y+2*c1/(sqrt(c2+2*y)))))/2;
  return(sqrt(r2));
}

double Omeq(double req)
{
  /*
    Omeq: Solves for Omega_eq given Req
  */
  double t1,t2;
  t1 = 6*Dl/(pow(req,8));
  t2 = 4*Cl/(pow(req,6));
  return(sqrt((t1-t2)/mass));
}

double leq(double req)
{
  /*
    leq: Solves for l=m*Omega*Req^2 given Req
  */
  double t1,t2;
  t1 = 6*Dl/(pow(req,7));
  t2 = 4*Cl/(pow(req,5));
  return(sqrt((t1-t2)*req*req*req*mass));
}

double h2Oh2(double r,double a,double b)
```

```
{
  /*
    h2Oh2: Evaluates the Potential Energy at (r,a,b)
  */
  double tQUAD,tDISP,tIND,tLJ;
  double tINDa,tINDb;
  double cb2,ca,s2b,r6;

  cb2 = pow(cos(a-b),2);
  ca = cos(a);
  s2b = sin(2*(a-b));
  r6 = pow(r,6);

  tQUAD = kv1*(ca*(3*cb2-1)-sin(a)*s2b)/pow(r,4);
  tDISP = -1*(kv2*cb2+kv3)/r6;
  tINDa = -1*kv4*(3*ca*ca+1)/r6;
  tINDb = -1*kv5*(cos(2*(a-b))*(3*ca*ca+cos(2*a))+2*sin(2*a)*s2b)/r6;
  tIND = tINDa+tINDb;
  tLJ = CLJ/(r6*r6);
  return(tQUAD+tDISP+tIND+tLJ);
}

double hamH2OH2(double x[6], double epsilon)
{
  /*
    ham: Evaluates the Hamiltonian at point x[6].
  */
  double h2Oh2(double R, double a, double b);
  double H,denom;
  H =
x[3]*x[3]*x[0]*x[0]*Ia*Ib+(epsilon-x[4])*(epsilon-x[4])*Ia*Ib+(x[4]-x[5])*(x[4]-x[5])*x[0]
*x[0]*mass*Ib+x[5]*x[5]*x[0]*x[0]*mass*Ia;
  denom = 2*mass*x[0]*x[0]*Ia*Ib;
  H = H/denom;
  H += h2Oh2(x[0],x[1],x[2]);
  return(H);
}

void ffieldH2OH2(double t, double *x, int n, double *f,double epsilon)
{
  /*
    ffield: evaluates ham. vector field
  */
  double ca,ca2,c2a,cb,cb2,c2b,sa,sa2,s2a,sb,sb2,s2b,r,pr,pa,pb,r2,r3,r4,r5,r6,r7;
  ca = cos(x[1]);
  ca2 = ca*ca;
  c2a = cos(2*x[1]);
  cb = cos(x[1]-x[2]);
  cb2 = cb*cb;
  c2b = cos(2*(x[1]-x[2]));
  sa = sin(x[1]);
  sa2 = sa*sa;
  s2a = sin(2*x[1]);
  sb = sin(x[1]-x[2]);
  sb2 = sb*sb;
  s2b = sin(2*(x[1]-x[2]));
  r = x[0];
  r2 = pow(r,2);
```

```
  r4 = pow(r2,2);
  r6 = pow(r2,3);
  r3 = r*r2;
  r5 = r*r4;
  r7 = r*r6;
  pr = x[3];
  pa = x[4];
  pb = x[5];
  f[0] = pr/mass;
  f[1] = (pa-pb)/Ia-(epsilon-pa)/(mass*r*r);
  f[2] = -(pa-pb)/Ia+pb/Ib;
  f[3] =
12*CLJ/pow(r,13)+pow(epsilon-pa,2)/(mass*r3)-6*(kv3+kv2*cb2)/r7-6*kv4*(1+3*ca2)/r7+3*Q*mu*
(ca*(-1+3*cb2)-sa*s2b)/r5-6*kv5*((3*ca2+c2a)*c2b+2*s2a*s2b)/r7;
  f[4] =
-6*kv4*ca*sa/r6-2*kv2*cb*sb/r6-(kv1*((1-3*cb2)*sa-2*c2b*sa-6*ca*cb*sb-ca*s2b))/r4+kv5*(c2b
*(-6*ca*sa-2*s2a)+4*c2b*s2a+4*c2a*s2b-2*(3*ca2+c2a)*s2b)/r6;
  f[5] =
2*kv2*cb*sb/r6-kv1*(2*c2b*sa+6*ca*cb*sb)/r4+kv5*(-4*c2b*s2a+2*(3*ca2+c2a)*s2b)/r6;
}

void bfieldH2OH2(double t, double *x,int n,double *f,double epsilon)
{
  /*
     bfield: Evaluates the negative ham. vector field
  */
  double ca,ca2,c2a,cb,cb2,c2b,sa,sa2,s2a,sb,sb2,s2b,r,pr,pa,pb,r2,r3,r4,r5,r6,r7;
  ca = cos(x[1]);
  ca2 = ca*ca;
  c2a = cos(2*x[1]);
  cb = cos(x[1]-x[2]);
  cb2 = cb*cb;
  c2b = cos(2*(x[1]-x[2]));
  sa = sin(x[1]);
  sa2 = sa*sa;
  s2a = sin(2*x[1]);
  sb = sin(x[1]-x[2]);
  sb2 = sb*sb;
  s2b = sin(2*(x[1]-x[2]));
  r = x[0];
  r2 = pow(r,2);
  r4 = pow(r2,2);
  r6 = pow(r2,3);
  r3 = r*r2;
  r5 = r*r4;
  r7 = r*r6;
  pr = x[3];
  pa = x[4];
  pb = x[5];
  f[0] = -1*(pr/mass);
  f[1] = -1*((pa-pb)/Ia-(epsilon-pa)/(mass*r*r));
  f[2] = -1*(-(pa-pb)/Ia+pb/Ib);
  f[3] =
-1*(12*CLJ/pow(r,13)+pow(epsilon-pa,2)/(mass*r3)-6*(kv3+kv2*cb2)/r7-6*kv4*(1+3*ca2)/r7+3*Q
*mu*(ca*(-1+3*cb2)-sa*s2b)/r5-6*kv5*((3*ca2+c2a)*c2b+2*s2a*s2b)/r7);
  f[4] =
-1*(-6*kv4*ca*sa/r6-2*kv2*cb*sb/r6-(kv1*((1-3*cb2)*sa-2*c2b*sa-6*ca*cb*sb-ca*s2b))/r4+kv5*
(c2b*(-6*ca*sa-2*s2a)+4*c2b*s2a+4*c2a*s2b-2*(3*ca2+c2a)*s2b)/r6);
```

```
  f[5] =
-1*(2*kv2*cb*sb/r6-kv1*(2*c2b*sa+6*ca*cb*sb)/r4+kv5*(-4*c2b*s2a+2*(3*ca2+c2a)*s2b)/r6);
}

double pRmax(double x0[6], double gam)
{
  /*
    initial guess for how far from fixed point var pR must be varied to hit e-surface
  */
  double ham(double x[6],double gamma);
  double h;
  h = fabs(ham(x0,gam)-heng)+fabs(x0[3]*x0[3]/(2*mass));
  return(sqrt(2*mass*h));
}

double pamin(double x0[6],double gam)
{
  /*
    initial guess for how far from fixed point var pa must be varied to hit e-surface
  */
  double ham(double x[6],double gamma);
  double h,b,c,discrim,bnum,cnum,denom;
  h = fabs(ham(x0,gam)-heng)+fabs((((gam-x0[4])*(gam-x0[4]))/(2*mass*x0[0]*x0[0])) +
((x0[4]-x0[5])*(x0[4]*x0[5])/(2*Ia)));
  bnum = -2*(Ia*gam+mass*x0[0]*x0[0]*x0[5]);
  cnum = Ia*gam*gam+mass*x0[0]*x0[0]*(x0[5]*x0[5]-2*Ia*h);
  denom = Ia+mass*x0[0]*x0[0];
  b = bnum/denom;
  c = cnum/denom;
  discrim = b*b-4*c;
  if(discrim<0) return(0);
  else return((-b-sqrt(discrim))/2);
}

double pamax(double x0[6],double gam)
{
  /*
    initial guess for how far from fixed point var pa must be varied to hit e-surface
  */
  double ham(double x[6],double gamma);
  double h,b,c,discrim,bnum,cnum,denom;
  h = fabs(ham(x0,gam)-heng)+fabs((((gam-x0[4])*(gam-x0[4]))/(2*mass*x0[0]*x0[0])) +
((x0[4]-x0[5])*(x0[4]*x0[5])/(2*Ia)));
  bnum = -2*(Ia*gam+mass*x0[0]*x0[0]*x0[5]);
  cnum = Ia*gam*gam+mass*x0[0]*x0[0]*(x0[5]*x0[5]-2*Ia*h);
  denom = Ia+mass*x0[0]*x0[0];
  b = bnum/denom;
  c = cnum/denom;
  discrim = b*b-4*c;
  if(discrim<0) return(0);
  else return((-b+sqrt(discrim))/2);
}

double pbmin(double x0[6],double gam)
{
  /*
    initial guess for how far from fixed point var pb must be varied to hit e-surface
  */
```

```
  double ham(double x[6],double gamma);
  double h,b,c,discrim,bnum,cnum,denom;
  h = fabs(ham(x0,gam)-heng)+ fabs((x0[5]*x0[5]/(2*Ib)) +
((x0[4]-x0[5])*(x0[4]*x0[5])/(2*Ia)));
  bnum = -2*Ib*x0[4];
  cnum = Ib*(x0[4]*x0[4]-2*Ia*h);
  denom = Ia+Ib;
  b = bnum/denom;
  c = cnum/denom;
  discrim = b*b-4*c;
  if(discrim<0) return(0);
  else return((-b-sqrt(discrim))/2);
}

double pbmax(double x0[6],double gam)
{
  /*
    initial guess for how far from fixed point var pb must be varied to hit e-surface
  */
  double ham(double x[6],double gamma);
  double h,b,c,discrim,bnum,cnum,denom;
  h = fabs(ham(x0,gam)-heng)+ fabs((x0[5]*x0[5]/(2*Ib)) +
((x0[4]-x0[5])*(x0[4]*x0[5])/(2*Ia)));
  bnum = -2*Ib*x0[4];
  cnum = Ib*(x0[4]*x0[4]-2*Ia*h);
  denom = Ia+Ib;
  b = bnum/denom;
  c = cnum/denom;
  discrim = b*b-4*c;
  if(discrim<0) return(0);
  else return((-b+sqrt(discrim))/2);
}

/*-------------------------------------*/
// TEST/DIAGNOSTIC FUNCTIONS:
//   FUNCTIONS:
//   oscillator(t,x[6],n,xdot[6]) - Vector Field for Harmonic Oscillator
//   testrand() - Tests the random number generator for mean and variance
//   VARIABLES:
//   x[6] - phase space point
//   xdot[6] - value of vector field at x[6]
//   freq[3] - frequencies of harmonic oscillator test system
//   t,n - (unused) variables for time (t) or condition (n) dependant vector fields
/*-------------------------------------*/

void oscillator(double t,double *x,int n,double *xdot,double freq[3])
{
  /*
     oscillator:  Evaluates the Ham. VF for a harmonic oscillator with frequencies
                    freq (for use testing integrator, etc.)
  */
  xdot[0] = freq[0]*x[3];
  xdot[1] = freq[1]*x[4];
  xdot[2] = freq[2]*x[5];
  xdot[3] = -freq[0]*x[0];
  xdot[4] = -freq[1]*x[1];
  xdot[5] = -freq[2]*x[2];
}
```

```
void testrand()
{
  /*
     testrand: Tests random number generator mean & variance for 999,999 points
  */
  int i;
  float r,mean=0,variance=0;
  for(i=0;i<999999;i++){
  r = rand();
  r = (r/RAND_MAX);
  mean = mean+r;
  variance = variance+(r-.5)*(r-.5);
  };
  printf("MEAN: %4.4f   VARIANCE: %4.4f\n",mean/i,variance/i);
}

/*-------------------------------------*/
// FUNCTIONS THAT REMAIN UNCHANGED WITH NEW MODELS:
//   FUNCTIONS:
//   roughbox(x0[6],min[5],max[5],stepmin,stepmax,intervals,resmax,resmin,epsilon)
//      - Determines rough bounding box at fixed point cut (fpcut)
//   refinebox(min[5],max[5],incut,cut,outcut,epsilon)
//      - Refines rough bounding box
//   refineboxB(min[5],max[5],incut,cut,outcut,epsilon)
//      - Refines rough bounding box for backward integrated LD
//
newcutbox(omin[5],omax[5],min[5],max[5],originalcut,newcut,opposingcut,points,epsilon)
//      - takes bounding box at originalcut and moves it to newcut through reverse
integration
//
newcutboxB(omin[5],omax[5],min[5],max[5],originalcut,newcut,opposingcut,points,epsilon)
//      - takes bounding box at originalcut and moves it to newcut through integration
//   randompoint(x[6],min[5],max[5],cut)
//      - Selects a point at random from box [min,max] with x[0] = cut
//   randomboundpoint(x[5],min[5],minT[5],max[5],maxT[5],cut)
//      - Finds point at random from boundary [minT-min,max-maxT]
//   energyfit(x[6],tol,e_0,pos,res,bound,epsilon)
//      - Projects x[6] onto e-surface e_0 using pos as dir. to proj.
//   energyfit3(x[6],min[5],max[5],epsilon)
//      - Attempts to proj. x[6] onto e-surface using any momentum direction needed
//   energyfitdown(x[6],min[5],max[5],fit,epsilon)
//      - currently unused projection function
//   VARIABLES:
//   x[6] - phase space point
//   min[5] - Minimum dimensions for bounding box (i.e., box dimensions are [min,max][5])
//   max[5] - Maximum dimensions for bounding box (i.e., box dimensions are [min,max][5])
//   minT[5] - Min. dimensions for test box surrounding bounding box
//   maxT[5] - Max. dimensions for test box surrounding bounding box
//   cut - the value of x[0]
//   tol - how close to energy surface projected point must be
//   e_0 - Energy surface to project to
//   pos - variable being projected
//   res - How many steps to take between x[pos] and bound
//   bound - the upper or lower bound opposite x[pos]
//   fit - Indicates whether energyfitdown successfully fit x[6] to energy surface
//   eps - system parameter (angular momentum, field strength, etc.)
/*-------------------------------------*/
```

```
void roughbox(double x0[6],double min[5],double max[5], double stepmin,double stepmax,
int intervals, int resmax,int resmin, double epsilon)
{
/*
    roughbox estimates the min & max dimensions of a box bounding incoming trajectories
by
    varying each parameter from the fixed point x0 until they pass the desired energy
level.
*/
    double ham(double x[6],double epsilon);
    int energyfit(double x[6],double tol,double e_0,int pos,int res,double bound,double
epsilon);
    int var,i,up;
    double h,h0,step,stepscale,bound,fit_tol,x[6];
    h0 = ham(x0,epsilon);
    h = heng - h0; //free energy of incoming trajectories
    if(h/resmax<1.e-5) fit_tol = h/resmax;
    else fit_tol = 1.e-5;
    //vary every coordinate that is not fixed; i.e., not x[0]=fpcut
    for(var=1;var<6;var++) {
       //vary every coordinate up & down from fixed point
       for(up=0;up<2;up++) {
          stepscale = pow(stepmax/stepmin,1./intervals); //each step is bigger than last
          for(i=0;i<6;i++) x[i] = x0[i];
          step = stepmin;
          if(up==0) step = -1*step;
          while(((ham(x,epsilon)-h0)<(h/resmax))||((ham(x,epsilon)-h0)>(h/resmin))) {
               if((ham(x,epsilon)-h0)>(h/resmin)) {
                 step = step/stepscale;
                 stepscale = (1+stepscale)/2;
               }
               step = step*stepscale;
               x[var] = x0[var] + step;
          } //stepping sequence stops short of energy surface in (h/resmin,h/resmax)
          i=1;
          //keep stepping with step until we cross energy surface
          while((ham(x,epsilon)-heng)<0) {
             x[var] = x0[var] + step*i;
             i++;
          }
          //use energyfit to project onto energy surface as usual
          if(up==0) {
             x[var] = x0[var] + step*i;
             bound = x0[var] + step*(i-1);
          }
          else {
             x[var] = x0[var] + step*(i-1);
             bound = x0[var] + step*i;
          }
          energyfit(x,1.e-15,heng,var,5,bound,epsilon);
          if(up==0) min[var-1] = x[var];
          if(up==1) max[var-1] = x[var];
       }
    }
}

void refinebox(double min[5],double max[5],double incut,double cut,double outcut,double
```

```
epsilon) {
/*
   refinebox expands the roughbox & newcutbox estimates until they minimally contain
   the entire compact intersection of the energy surface.

   Refinement is achieved by increasing the existing box by a factor %grow% and testing
   points in the boundary between the new and old box.  If the point projects onto
   the energy surface and has the desired properties, then we expand the original box
   to contain this point.  Repeat until 500 consecutive points are sample without the
   desired properties.  Also, use a number of factors %grow% to speed up process.
*/
   void randompoint(double x[6],double min[5],double max[5],double cut);
   int randomboundpoint(double x[6],double min[5],double minT[5],double max[5],double
maxT[5],double cut);
   int energyfit3(double x[6],double tol,double min[5],double max[5],double epsilon);
   double minT[5],maxT[5],diff[5],mintemp[5],maxtemp[5];
   int i,j,k;
   int firstrun,firstcount,firstflag;
   int badpoints,rpoints,fit,var,plus;
   double randx[6],randxs[6],grow;
   double t,h,hmin,hmax;
   FILE *ferror;
   //giving original box a 10% border
   for(j=0;j<5;j++) {
      diff[j] = max[j]-min[j];
      minT[j] = min[j]-.05*diff[j];
      maxT[j] = max[j]+.05*diff[j];
   }
   //Growing box around reactive trajectories
   grow = 0;
   firstrun = 1;
   firstcount = 1;
   firstflag = 0;
   for(i=0;i<10;i++) {
      if(i==0) grow = .20;
      if(i==1) grow = .16;
      if(i==2) grow = .12;
      if(i==3) grow = .08;
      if(i==4) grow = .07;
      if(i==5) grow = .06;
      if(i==6) grow = .05;
      if(i==7) grow = .04;
      if(i==8) grow = .03;
      if(i==9) grow = .02;
      printf("(%24.16e)i = %d\n",epsilon,i);
      badpoints = 0;
      fit = 1;
      if(i==9) {
        for(j=0;j<5;j++) {
           diff[j] = fabs(max[j] - min[j]);
           if(diff[j]<=1.e-6) firstflag = 1;
        }
      }
      //sample in boundary until 500 bad points are consecutively sampled
      while(badpoints<500) {
           if((badpoints==0)&&(firstrun!=1)) {
             for(j=0;j<5;j++) {
                diff[j] = max[j] - min[j];
```

```
                minT[j] = min[j]-grow*diff[j];
                maxT[j] = max[j]+grow*diff[j];
              }
            }
            //sample in boundary
            var = randomboundpoint(randx,min,minT,max,maxT,cut);
            if(var<0) {
              var = -1*var;
              plus = 0;
            }
            else plus = 1;
            for(j=0;j<5;j++) {
               maxtemp[j] = maxT[j];
               mintemp[j] = minT[j];
            }
            if(plus==0) {
              mintemp[var-1] = minT[var-1];
              maxtemp[var-1] = min[var-1];
            }
            if(plus==1) {
              mintemp[var-1] = max[var-1];
              maxtemp[var-1] = maxT[var-1];
            }
            //project in boundary using mintemp and maxtemp
            fit = energyfit3(randx,1.e-12,mintemp,maxtemp,epsilon);
            if(firstrun==1) {
              firstcount++;
              rpoints = 0;
              while(rpoints==0) {
                  randompoint(randx,minT,maxT,cut);
                  rpoints = energyfit3(randx,1.e-12,minT,maxT,epsilon);
              }
              fit = rpoints;
              if(firstcount>=2000) {
                badpoints = 1000;
                i = 10;
                firstflag = 1;
                ferror=fopen("./fullscatter/ERROR","a");
                if(ferror==NULL) {printf("refinebox: cannot open file
%s!\n","./fullscatter/ERROR"); exit(1);}
                fprintf(ferror,"ERROR: epsilon = %e, Still on firstcount at INS,
CUT!\n",epsilon);
                fclose(ferror);
              }
            }
            if(fit==0) badpoints++;
            //integrate point to see if it is reactive
            if(fit!=0) {
              t=0.e0;
              h=1.e-1;
              hmin=1.e-3;
              hmax = 1.e+1;
              for(k=0;k<6;k++) randxs[k] = randx[k];
              while((randx[0]<outcut)&&(randx[0]>incut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,ffield,epsilon);
              if(randx[0]>=outcut) {
                badpoints++;
              }
```

```
            if(randx[0]<=incut) {
              if(firstrun==1) firstrun = 0;
              badpoints = 0;
              for(k=1;k<6;k++) {
                 if(randxs[k]<min[k-1]) min[k-1] = randxs[k];
                 if(randxs[k]>max[k-1]) max[k-1] = randxs[k];

              }
            }
          }
      }
   }
   //EXPANDING BOX BY 1 PERCENT (JUST TO BE SAFE)
   for(i=0;i<5;i++) {
      diff[i] = max[i]-min[i];
      min[i] = min[i]-.01*diff[i];
      max[i] = max[i]+.01*diff[i];
      printf("(eps=%le)min = %le   max = %le\n",epsilon,min[i],max[i]);
   }
}

void refineboxB(double min[5],double max[5],double incut,double cut,double outcut,double
epsilon) {
/*
   refineboxB expands the roughbox & newcutboxB estimates until they minimally contain
   the entire compact intersection of the energy surface.

   Refinement is essentially the same as in refinebox, except this uses a backward
   integration to determine if point is /reverse/ reactive
*/
   void randompoint(double x[6],double min[5],double max[5],double cut);
   int randomboundpoint(double x[6],double min[5],double minT[5],double max[5],double
maxT[5],double cut);
   int energyfit3(double x[6],double tol,double min[5],double max[5],double epsilon);
   double minT[5],maxT[5],diff[5],mintemp[5],maxtemp[5];
   int i,j,k;
   int firstrun,firstcount,firstflag;
   int badpoints,rpoints,fit,var,plus;
   double randx[6],randxs[6],grow;
   double t,h,hmin,hmax;
   FILE *ferror;
   for(j=0;j<5;j++) {
      diff[j] = max[j]-min[j];
      minT[j] = min[j]-.05*diff[j];
      maxT[j] = max[j]+.05*diff[j];
   }
   //GROWING BOX AROUND INCOMING TRAJECTORIES
   grow = 0;
   firstrun = 1;
   firstcount = 1;
   firstflag = 0;
   for(i=0;i<10;i++) {
      if(i==0) grow = .20;
      if(i==1) grow = .16;
      if(i==2) grow = .12;
      if(i==3) grow = .08;
      if(i==4) grow = .07;
      if(i==5) grow = .06;
```

```
      if(i==6) grow = .05;
      if(i==7) grow = .04;
      if(i==8) grow = .03;
      if(i==9) grow = .02;
      printf("(%24.16e)i = %d\n",epsilon,i);
      badpoints = 0;
      fit = 1;
      if(i==9) {
        for(j=0;j<5;j++) {
           diff[j] = fabs(max[j] - min[j]);
           if(diff[j]<=1.e-6) firstflag = 1;
        }
      }
      while(badpoints<500) {
           if((badpoints==0)&&(firstrun!=1)) {
             for(j=0;j<5;j++) {
                diff[j] = max[j] - min[j];
                minT[j] = min[j]-grow*diff[j];
                maxT[j] = max[j]+grow*diff[j];
             }
           }
           var = randomboundpoint(randx,min,minT,max,maxT,cut);
           if(var<0) {
             var = -1*var;
             plus = 0;
           }
           else plus = 1;
           for(j=0;j<5;j++) {
              maxtemp[j] = maxT[j];
              mintemp[j] = minT[j];
           }
           if(plus==0) {
             mintemp[var-1] = minT[var-1];
             maxtemp[var-1] = min[var-1];
           }
           if(plus==1) {
             mintemp[var-1] = max[var-1];
             maxtemp[var-1] = maxT[var-1];
           }
           fit = energyfit3(randx,1.e-12,mintemp,maxtemp,epsilon);
           if(firstrun==1) {
             firstcount++;
             rpoints = 0;
             while(rpoints==0) {
                 randompoint(randx,minT,maxT,cut);
                 rpoints = energyfit3(randx,1.e-12,minT,maxT,epsilon);
             }
             fit = rpoints;
             if(firstcount>=2000) {
               badpoints = 1000;
               i = 10;
               firstflag = 1;
               ferror=fopen("./fullscatter/ERROR","a");
               if(ferror==NULL) {printf("refinebox: cannot open file
%s!\n","./fullscatter/ERROR"); exit(1);}
               fprintf(ferror,"ERROR: epsilon = %e, Still on firstcount at INS,
CUT!\n",epsilon);
               fclose(ferror);
```

```
            }
          }
          if(fit==0) badpoints++;
          if(fit!=0) {
            t=0.e0;
            h=1.e-1;
            hmin=1.e-3;
            hmax = 1.e+1;
            for(k=0;k<6;k++) randxs[k] = randx[k];
            while((randx[0]<outcut)&&(randx[0]>incut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,bfield,epsilon);
            if(randx[0]>=outcut) {
              badpoints++;
            }
            if(randx[0]<=incut) {
              if(firstrun==1) firstrun = 0;
              badpoints = 0;
              for(k=1;k<6;k++) {
                 if(randxs[k]<min[k-1]) min[k-1] = randxs[k];
                 if(randxs[k]>max[k-1]) max[k-1] = randxs[k];

              }
            }
          }
      }
   }
   //EXPANDING BOX BY 1 PERCENT
   for(i=0;i<5;i++) {
      diff[i] = max[i]-min[i];
      min[i] = min[i]-.01*diff[i];
      max[i] = max[i]+.01*diff[i];
      printf("(eps=%le)min = %le   max = %le\n",epsilon,min[i],max[i]);
   }
}

void newcutbox(double omin[5],double omax[5],double min[5],double max[5],double
originalcut, double newcut, double opposingcut,int points,double epsilon) {
/*
   newcutbox takes a bounding box at original cut, samples points from the box, and
   integrates them either forward or backward to newcut to obtain a rough bounding
   box at newcut.

   forward and backward integration is chosen by whether or not newcut-opposingcut < 0

   points that integrate to opposingcut are discarded
*/
   void randompoint(double x[6],double min[5],double max[5],double cut);
   int energyfit3(double x[6],double tol,double min[5],double max[5],double epsilon);
   int i,j,ins,outs,rpoints,newdir;
   double t,h,hmin,hmax;
   double randx[6],randxs[6];

   //chooses direction of integration
   if((newcut-opposingcut)>0) newdir = 1;
   else if((newcut-opposingcut)<0) newdir = -1;
   outs = 0;
   for(i=0;i<5;i++) {
      omin[i] = 1.e+10;
```

```
      omax[i] = -1.e+10;
   }
   while(outs<points) {
        t=0.e0;
        h=1.e-2;
        hmin=1.e-3;
        hmax = 1.e-1;
        rpoints = 0;
        //select points on e-surface from roughbox
        while(rpoints==0) {
             randompoint(randx,min,max,originalcut);
             rpoints = energyfit3(randx,1.e-15,min,max,epsilon);
        }
        for(j=0;j<6;j++) randxs[j] = randx[j];
        if(newdir==1) {
          while((randx[0]<newcut)&&(randx[0]>opposingcut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,bfield,epsilon);
          if(randx[0]>=newcut) {
            for(j=1;j<6;j++) {
               //expand box at newcut to contain new trajectory
               if(randx[j]<omin[j-1]) omin[j-1] = randx[j];
               if(randx[j]>omax[j-1]) omax[j-1] = randx[j];
            }
            outs++;
          }
          if(randx[0]<=opposingcut) ins++;
        }
        if(newdir==-1) {
          while((randx[0]>newcut)&&(randx[0]<opposingcut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,ffield,epsilon);
          if(randx[0]<=newcut) ins++;
          if(randx[0]>=opposingcut) {
           for(j=1;j<6;j++) {
               //expand box at newcut to contain new trajectory
               if(randx[j]<omin[j-1]) omin[j-1] = randx[j];
               if(randx[j]>omax[j-1]) omax[j-1] = randx[j];
            }
            outs++;
          }
        }
   }
}

void newcutboxB(double omin[5],double omax[5],double min[5],double max[5],double
originalcut, double newcut, double opposingcut,int points,double epsilon) {
/*
   newcutboxB is the same as newcutbox, except the direction of integration is reversed
*/
   void randompoint(double x[6],double min[5],double max[5],double cut);
   int energyfit3(double x[6],double tol,double min[5],double max[5],double epsilon);
   int i,j,ins,outs,rpoints,newdir;
   double t,h,hmin,hmax;
   double randx[6],randxs[6];

   if((newcut-opposingcut)>0) newdir = 1;
   else if((newcut-opposingcut)<0) newdir = -1;
   outs = 0;
   for(i=0;i<5;i++) {
```

```
      omin[i] = 1.e+10;
      omax[i] = -1.e+10;
   }
   while(outs<points) {
        t=0.e0;
        h=1.e-2;
        hmin=1.e-3;
        hmax = 1.e-1;
        rpoints = 0;
        while(rpoints==0) {
             randompoint(randx,min,max,originalcut);
             rpoints = energyfit3(randx,1.e-15,min,max,epsilon);
        }
        for(j=0;j<6;j++) randxs[j] = randx[j];
        if(newdir==1) {
          while((randx[0]<newcut)&&(randx[0]>opposingcut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,ffield,epsilon);
          if(randx[0]>=newcut) {
            for(j=1;j<6;j++) {
               if(randx[j]<omin[j-1]) omin[j-1] = randx[j];
               if(randx[j]>omax[j-1]) omax[j-1] = randx[j];
            }
            outs++;
          }
          if(randx[0]<=opposingcut) ins++;
        }
        if(newdir==-1) {
          while((randx[0]>newcut)&&(randx[0]<opposingcut))
rk78(&t,randx,&h,1.e-10,hmin,hmax,6,ffield,epsilon);
          if(randx[0]<=newcut) ins++;
          if(randx[0]>=opposingcut) {
           for(j=1;j<6;j++) {
               if(randx[j]<omin[j-1]) omin[j-1] = randx[j];
               if(randx[j]>omax[j-1]) omax[j-1] = randx[j];
            }
            outs++;
          }
        }
   }
}

void randompoint(double x[6],double min[5],double max[5],double cut)
{
/*
    x[6]: point randomly selected in box [min,max] with x[0] = cut.
*/
    int i;
    double r;
    x[0] = cut;
    for(i=1;i<6;i++)
    {
       r=rand();
       r=r/RAND_MAX;
       x[i] = min[i-1]+r*(max[i-1]-min[i-1]);
    }
}

int randomboundpoint(double x[6],double min[5],double minT[5],double max[5],double
```

```
maxT[5],double cut)
{
/*
     x[6]: point between box [min,max] and box [minT,maxT]
     step 1 - x[0] = cut
     step 2 - One of x[1]-x[5] is chosen at random, call it %x%
     step 3 - %x% is set either between (minT,min) or between (max,maxT)
     step 4 - The remaining four coordinates are chosen at random between (minT,maxT)

     Return var, the boundary coordinate.
*/
     int i,dim = 6,var;
     double r;
     //step 1
     x[0] = cut;
     //step 2
     r = rand();
     r = (dim-1)*r/RAND_MAX;
     var = 0;
     for(i=0;i<dim-1;i++) {
        if((r>i)&&(r<i+1)) var = i+1;
     }
     //step 3
     r = rand();
     r = r/RAND_MAX;
     if(r<.5) var = -1*var; //x[var] will be in max_boundary
     if(r>=.5) var = var;   //x[var] will be in min_boundary
     //step 4
     for(i=1;i<dim;i++) {
        r = rand();
        r = r/RAND_MAX;
        if(i!=fabs(var)) x[i] = minT[i-1]+r*(maxT[i-1]-minT[i-1]);
        if(i==fabs(var)) {
          if(var<0) x[i] = minT[i-1]+r*(min[i-1]-minT[i-1]);
          if(var>0) x[i] = maxT[i-1]+r*(max[i-1]-maxT[i-1]);
        }
     }
     return(var);
}


int energyfit(double x[6],double tol,double e_0,int pos,int res,double bound,double
epsilon)
{
/*
    x[6] : Point projected onto energy surface H=e_0 via momentum variable x[pos].
    tol  : Tolerance for how close the energy of point *x must be to the surface e_0.
    res  : How many steps to take between x[pos] and bound.
    bound: Bound is the upper (or lower) bound for energyfit.
           x[pos] should be the lower (or upper) bound.
           bound and x[pos] should be determined before calling energyfit.
    Essentially, I take /res/ steps from x[pos] to bound and wait for the point to cross
the
      energy surface.  If the point doesn't cross the surface, then energyfit returns 0,
      indicating that it failed to fit a point to the surface.  If the point does cross,
then I
      set res=2 and repeat until I refine the crossing point so that the energy of *x is
within
```

```
      tolerance of e_0.
*/
    double ham(double x[6],double epsilon);
    int i,j,fit,exitval,firstrun;
    double error,y[6],range;
    fit = 0;
    error = 1.;
    range = bound-x[pos];
    exitval=0;
    firstrun = 1;
    while((error>tol)&&(fit!=-1))
    {
         if(firstrun==0) res = 2;
         exitval = 0;
         i=0;
         while(exitval==0)
         {
             for(j=0;j<6;j++) y[j] = x[j];
             x[pos] = x[pos]+(range/res);
             if((ham(x,epsilon)>e_0)&&(ham(y,epsilon)<e_0)) {exitval = 1; fit = 1;}
             if((ham(x,epsilon)<e_0)&&(ham(y,epsilon)>e_0)) {exitval = 1; fit = 1;}
             if((i==res)&&(exitval==0)) {exitval = 1; fit = -1;}
             i++;
         }
         firstrun = 0;
         range = x[pos]-y[pos];
         error = fabs(ham(x,epsilon)-e_0);
         for(j=0;j<6;j++) x[j] = y[j];
    }
    if(fit!=1) fit=0;
    return(fit);
}

int energyfit3(double x[6],double tol,double min[5],double max[5],double epsilon)
{
/*
   x[6]: A point fit on the energy surface between [min,max].
   min-max: Must be chosen ahead of time to reflect which variable is on the boundary.
   energyfit3 attempts to project point onto energy surface in [min,max] using
     all momentum variables, projecting up or down, if need be.
   energyfit3 returns 0 if fit is unsuccessful
*/
   int i,j,exit;
   int code[3],swapc,up;
   double r[3],rnd,swapr,bound;
   double x0[6];
   bound = 0; //so bound is not "possibly uninitialized"
   //step 1 - Decide whether to project up or down
   rnd = rand();
   rnd = rnd/RAND_MAX;
   if(rnd<.5) up = 0;
   else up = 1;
   //step 2 - Choose order of momentum variables to try projecting with
   for(i=0;i<3;i++) {
      code[i] = i;
      r[i] = rand();
      r[i] = r[i]/RAND_MAX;
   }
```

```
   for(i=0;i<2;i++) {
      for(j=i+1;j<3;j++) {
         if(r[i]<r[j]) {
           swapc = j;
           swapr = r[j];
           r[j] = r[i];
           r[i] = swapr;
           code[j] = code[i];
           code[i] = swapc;
         }
      }
   }
   //step 3a - Try projecting using mom. vars. decided in step 2
   //         with up or down decided in step 1.
   exit=0;
   i=0;
   while((exit!=1)&&(i<3)) {
        for(j=0;j<6;j++) x0[j] = x[j];
        if(up==0) {
          bound = min[code[i]+2];
          x0[i+3] = max[code[i]+2];
        }
        if(up==1) {
          bound = max[code[i]+2];
          x0[i+3] = min[code[i]+2];
        }
        exit=energyfit(x0,tol,heng,code[i]+3,5,bound,epsilon);
        i++;
   }

   //step 3b - If the "up" projection doesnt work, then try
   //         "!up" projection so as not to be biased.
   if(exit!=1) {
     i=0;
     while((exit!=1)&&(i<3)) {
          for(j=0;j<6;j++) x0[j] = x[j];
          if(up==0) {
            bound = max[code[i]+2];
            x0[i+3] = min[code[i]+2];
          }
          if(up==1) {
            bound = min[code[i]+2];
            x0[i+3] = max[code[i]+2];
          }
          exit=energyfit(x0,tol,heng,code[i]+3,5,bound,epsilon);
          i++;
     }
   }
   if(exit==1) {
     exit=i+2;
     for(i=0;i<6;i++) x[i] = x0[i];
   }
   return(exit);
}

int energyfitdown(double x[6],double min[5],double max[5],int fit,double epsilon)
{
/*
```

```
   x[6]: A point projected onto energy surface between [min,max]
   min-max: Must be chosen ahead of time to reflect which variable is on the boundary.
   fit: coordinate being fit
   energyfitdown returns 0 if fit is unsuccsesful
*/
   int exit;
   int up;
   double rnd,bound;
   rnd = rand();
   rnd = rnd/RAND_MAX;
   /*-choose randomly whether to project down or up-*/
   if(rnd<.5) up = 0;
   else up = 1;
        if(up==0) {
          bound = min[fit-1];
          x[fit] = max[fit-1];
        }
        if(up==1) {
          bound = max[fit-1];
          x[fit] = min[fit-1];
        }
        /*-project onto energy surface using /fit/-*/
        exit=energyfit(x,1.e-13,heng,fit,5,bound,epsilon);
   return(exit);
}
```

```
/*
this is a general purpose package to integrate ordinary differential
equations. the method used is based on two Runge-Kutta
algorithms of order 7 and 8 with automatic stepsize control.

NOTE: I inherited this code during my work with Dr. Frederic Gabern
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

static double alfa[13]={
           0.e0,      2.e0/27.e0,       1.e0/9.e0,        1.e0/6.e0,
     5.e0/12.e0,           0.5e0,       5.e0/6.e0,        1.e0/6.e0,
      2.e0/3.e0,       1.e0/3.e0,            1.e0,             0.e0,
           1.e0};

static double beta[79]={
           0.e0,      2.e0/27.e0,      1.e0/36.e0,       1.e0/12.e0,
     1.e0/24.e0,            0.e0,       1.e0/8.e0,       5.e0/12.e0,
           0.e0,    -25.e0/16.e0,     25.e0/16.e0,            .5e-1,
           0.e0,            0.e0,           .25e0,             .2e0,
  -25.e0/108.e0,            0.e0,            0.e0,    125.e0/108.e0,
   -65.e0/27.e0,    125.e0/54.e0,    31.e0/300.e0,             0.e0,
           0.e0,            0.e0,    61.e0/225.e0,       -2.e0/9.e0,
   13.e0/900.e0,            2.e0,            0.e0,             0.e0,
    -53.e0/6.e0,    704.e0/45.e0,    -107.e0/9.e0,      67.e0/90.e0,
           3.e0,   -91.e0/108.e0,            0.e0,             0.e0,
   23.e0/108.e0,  -976.e0/135.e0,    311.e0/54.e0,     -19.e0/60.e0,
     17.e0/6.e0,     -1.e0/12.e0, 2383.e0/4100.e0,             0.e0,
           0.e0,  -341.e0/164.e0, 4496.e0/1025.e0,    -301.e0/82.e0,
2133.e0/4100.e0,     45.e0/82.e0,    45.e0/164.e0,      18.e0/41.e0,
    3.e0/205.e0,            0.e0,            0.e0,             0.e0,
           0.e0,     -6.e0/41.e0,    -3.e0/205.e0,      -3.e0/41.e0,
     3.e0/41.e0,      6.e0/41.e0,            0.e0, -1777.e0/4100.e0,
           0.e0,            0.e0,  -341.e0/164.e0,  4496.e0/1025.e0,
  -289.e0/82.e0, 2193.e0/4100.e0,     51.e0/82.e0,     33.e0/164.e0,
    12.e0/41.e0,            0.e0,            1.e0};

static double c7[11]={
   41.e0/840.e0,            0.e0,            0.e0,             0.e0,
           0.e0,    34.e0/105.e0,      9.e0/35.e0,       9.e0/35.e0,
    9.e0/280.e0,     9.e0/280.e0,    41.e0/840.e0};

static double c8[13]={
           0.e0,            0.e0,            0.e0,             0.e0,
           0.e0,    34.e0/105.e0,      9.e0/35.e0,       9.e0/35.e0,
    9.e0/280.e0,     9.e0/280.e0,            0.e0,     41.e0/840.e0,
   41.e0/840.e0};

static double *x7,*x8,*xpon,*dx,*k[13];
static int neq=0;

#define MAX(a,b) (((a)<(b)) ? (b) : (a))
#define SGN(a) (((a)<0) ? -1 : 1)
#define NEG(a) (((a)<0) ? 1 : 0)
#define SGNCHG(y1,y2,x) (SGN(y1)*SGN(y2)>0 ? 0 : SGN(y2)*NEG(x))
```

```
void ini_rk78(int n)
/*
this is to allocate space for the package. it must be called before
calling the rk78 routine.

parameters:
n: dimension of the system to be integrated.
*/
{
   int j;
   if (n < 1) {puts("ini_rk78: n must be at least 1"); exit(1);}
   if (neq != 0)
      {
         free(x7);
         free(x8);
         free(xpon);
         free(dx);
         for (j=0; j<13; j++) free(k[j]);
      }
   neq=n;
   x7=(double*)malloc(n*sizeof(double));
   if (x7 == NULL) {puts("ini_rk78: out of memory (1)"); exit(1);}
   x8=(double*)malloc(n*sizeof(double));
   if (x8 == NULL) {puts("ini_rk78: out of memory (2)"); exit(1);}
   xpon=(double*)malloc(n*sizeof(double));
   if (xpon == NULL) {puts("ini_rk78: out of memory (3)"); exit(1);}
   dx=(double*)malloc(n*sizeof(double));
   if (dx == NULL) {puts("ini_rk78: out of memory (4)"); exit(1);}
   for (j=0; j<13; j++)
   {
      k[j]=(double*)malloc(n*sizeof(double));
      if (k[j] == NULL) {puts("ini_rk78: out of memory (5)"); exit(1);}
   }
   return;
}
void end_rk78(int n)
/*
this is to free the memory previously allocated by ini_rk78.

parameter:
n: dimension of the systems of odes. it should coincide with the value
   previously used by ini_rk78.
*/
{
   int j;
   if (n != neq) puts("end_rk78 warning: dimensions do not coincide!");
   free(x7);
   free(x8);
   free(xpon);
   free(dx);
   for (j=0; j<13; j++) free(k[j]);
   return;
}
int rk78(double *at, double x[], double *ah, double tol,
            double hmin, double hmax, int n,
            void (*deriv)(double, double *, int, double *,double),double epsilon)
/*
```

```
this routine performs one step of the integration procedure.
the initial condition (at,x) is changed by a new one corresponding
to the same orbit. the error is controlled by the threshold tol,
and an estimate of the error produced in the actual step is returned
as the value of the function.

parameters:
at:   time. input: time corresponding to the actual initial condition.
            output: new value corresponding to the new initial condition.
x:    position. same remarks as at.
ah:   time step (it can be modified by the routine according to the
      given threshold).
tol:  threshold to control the integration error.
hmin: minimun time step allowed.
hmax: maximum time step allowed.
n:    dimension of the system of odes.
deriv: function that returns the value of the vectorfield.

returned value: WHETHER OR NOT THE SIGN CHANGES!!! IN TERMS OF 0,+1,-1 !!!


an estimate of the error produced in the actual step of
integration.
*/
{
   double tpon,tol1,err,nor,kh,beth,h1;
   int i,j,l,m;
   if (n > neq) {printf("rk78: wrong dimension (%d and %d)\n",n,neq); exit(1);}
   do {
/*
      this is to compute the values of k
*/
      m=0;
      for (i=0; i<13; i++)
      {
         tpon=*at+alfa[i]*(*ah);
         for (j=0; j<n; j++ ) xpon[j]=x[j];
         for ( l=0; l<i; l++ )
         {
            ++m;
            beth=*ah*beta[m];
            for (j=0; j<n; j++) xpon[j] += beth*k[l][j];
         }
         (*deriv)(tpon,xpon,n,dx,epsilon);
         for (j=0; j<n; j++ ) k[i][j] = dx[j];
      }
/*
      this is to compute the rk7 and rk8 predictions
*/
      err=nor=0.e0;
      for (j=0; j<n; j++)
      {
         x7[j]=x8[j]=x[j];
         for (l=0; l<11; l++)
         {
            kh=*ah*k[l][j];
            x7[j] += kh*c7[l];
            x8[j] += kh*c8[l];
```

```
         }
         x8[j] += *ah*(c8[11]*k[11][j]+c8[12]*k[12][j]);
         err += fabs(x8[j]-x7[j]);
         nor += fabs(x8[j]);
      }
      err /= n;
/*
      next lines compute the new time step h
*/
      tol1=tol*(1+nor/100);
      if (err < tol1) err=MAX(err,tol1/256);
      h1=*ah;
      *ah*=0.9*pow(tol1/err,0.125);
      if (fabs(*ah) < hmin ) *ah=hmin*SGN(*ah);
      if (fabs(*ah) > hmax ) *ah=hmax*SGN(*ah);
   } while ((err >= tol1) && (fabs (*ah) > hmin));
   *at += h1;
/*
   the next line determines whether the trajectory has made an orbit
*/
   i=SGNCHG(x[1],x8[1],x[0]);
   for (j=0; j<n; j++) x[j]=x8[j];
   return (i);
}
```

D4 moduls.h - System Parameters

```
/*
   File where we define the system constants
*/

#define max(a,b) ((a)<(b) ? (b) : (a))
#define min(a,b) ((a)>(b) ? (b) : (a))

//RYDBERG CONSTANTS:
#define    PI                  3.1415926535897932
#define    heng                (-1.52000000)

//H2O-H2 CONSTANTS:
/*
static double mu        =       0.730;
static double Q         =       0.710;
#define Ia                      (PI*2414)
#define Ib                      (PI*576)
static double IPa       =       0.464;
static double IPb       =       0.567;
static double IP        =       (IPa*IPb/(IPa+IPb));
static double abarH2O   =       9.830;
static double alphpl    =       6.803;
static double alphpp    =       4.845;
static double dalph     =       (alphpl-alphpp);
static double A         =       (abarH2O*alphpl*IP/4);
static double B         =       (abarH2O*alphpp*IP/4);
static double abarH2    =       ((alphpl+alphpp)/2);
//static double Cl      (-3*mu*Q/4)
//static double Dl      (A+5*B+2*mu*mu*alphpp)
static double Cl        =       (-3*mu*Q/2);
static double Dl        =       (4*A+2*B+2*mu*mu*alphpl);
#define       CLJ               100.
#define       mass              1.8
#define       E0                (3.e-3)
//static double gamma   8.0
//TO MAKE POTENTIAL FASTER
static double kv1       =       (3*mu*Q/4);
static double kv2       =       (3*(A-B));
static double kv3       =       (A+5*B);
static double kv4       =       (mu*mu*abarH2/2);
static double kv5       =       (mu*mu*dalph/4);
*/
```

```
#
# Makefile to build up the several programs of the package.
#
# NOTE: I inherited this code during my work with Dr. Frederic Gabern
#
#
# in the next lines you have to choose the right parameters for your
# system. I've put as default values the ones corresponding to the GNU
# C/C++ compiler, but you should change them if you want to use a
# different compiler. I've added (in commented lines) the corresponding
# values for the compiler that comes with HP UX 10.20.
#
# Ansi C compiler
#
CC=gcc
# CC=cc (this is for HP UX 10.20)
#
# C++ compiler
#
CP=g++
CF=g77
# CP=CC (this is for HP UX 10.20)
#
# compilation flags for the C compiler
#
CFLAGS=-O3 -Wall
# CFLAGS=-O -Aa (this is for HP UX 10.20)
#
# compilation flags for the C++ compiler
#
CPPFLAGS=-O3 -Wall
# CPPFLAGS=-O (this is for HP UX 10.20)
#
# linking flags
#
LFLAGS=-lg2c -lm -s
#
# directories to store the binaries and to find/store data files resp.
# I've used relative pathnames, but it is better to use absolute ones
# (with the relative pathnames, you have to execute the programs from
# the BIN directory, otherwise they could not find the DATA directory).
#
BIN=./bin/
DAT=./data/
#
# directory to put working files (they will be erased at the end of
# the execution).
#
TMP=./
#
# ========================================================
#  you shouldn't need to modify anything beyond this line
# ========================================================
#
#######################################
# to build up the normal form program #
#######################################
RYD=newt.o main-ryd.o rk78.o
```

```
ryd: $(RYD)
	$(CP) $(CPPFLAGS) $(RYD) -o $(BIN)ryd $(LFLAGS)
	rm -f *.o


FS=newt.o fullscatter.o rk78.o

fs: $(FS)
	$(CP) $(CPPFLAGS) $(FS) -o $(BIN)fs $(LFLAGS)
	rm -f *.o

FSB=newt.o fullscatterB.o rk78.o

fsb: $(FSB)
	$(CP) $(CPPFLAGS) $(FSB) -o $(BIN)fsb $(LFLAGS)
	rm -f *.o

FSKEPE_COMPARE=newt.o fullscatterKEPE_COMPARE.o rk78.o

fskepe_compare: $(FSKEPE_COMPARE)
	$(CP) $(CPPFLAGS) $(FSKEPE_COMPARE) -o $(BIN)fskepe_compare $(LFLAGS)
	rm -f *.o

############################
# how to make the .o files #
############################
main-ryd.o: main-ryd.cc
	$(CP) -c $(CPPFLAGS) -DDATA=\"$(DAT)\" -DTEMP=\"$(TMP)\" main-ryd.cc
fullscatter.o: fullscatter.cc
	$(CP) -c $(CPPFLAGS) -DDATA=\"$(DAT)\" -DTEMP=\"$(TMP)\" fullscatter.cc
fullscatterB.o: fullscatterB.cc
	$(CP) -c $(CPPFLAGS) -DDATA=\"$(DAT)\" -DTEMP=\"$(TMP)\" fullscatterB.cc
fullscatterKEPE_COMPARE.o: fullscatterKEPE_COMPARE.cc
	$(CP) -c $(CPPFLAGS) -DDATA=\"$(DAT)\" -DTEMP=\"$(TMP)\"
fullscatterKEPE_COMPARE.cc
rk78.o: rk78.cc
	$(CP) -c $(CPPFLAGS) rk78.cc
newt.o: newt.cc
	$(CP) -c $(CFLAGS) newt.cc
#########
# clean #
#########
clean:
	rm -f *.o
	rm -f $(BIN)*
```

References

1. Dellnitz, M., Grubits, K.A., Marsden, J.E., Padberg, K., and B. Thiere. Set oriented computation of transport rates in 3-degree of freedom systems: scattering rates for the Rydberg atom in crossed fields. *in preparation*, [2005].

2. Faure, A., Wiesenfeld, L., Wernli, M., and P. Valiron. The role of rotation in the vibrational relaxation of water by hydrogen molecules. *J. Chem. Phys.* **123**:104309, [2005].

3. Faure, A., Valiron, P., Wernli, M., Wiesenfeld, L., Rist, C., Noga, J., and J. Tennyson. A full nine-dimensional potential-energy surface for hydrogen molecule-water collisions. *J. Chem. Phys.* **122**:221102 , [2005].

4. Gabern, F., Koon, W.S., Marsden, J.E., and S.D. Ross. Theory and Computation of Non-RRKM Lifetime Distributions and Rates in Chemical Systems with Three or More Degrees of Freedom. *in preparation*,[2005].

5. Jaffe, C., Kawai, S., Palacian, J., Yanguas, P., and T. Uzer. A new look at the transition state: Wigner's dynamical perspective revisited. *Geometric Structures of Phase Space in Multidimensional Chaos: A Special Volume of Advances in Chemical Physics, Part A* **130**, [2005].

6. Jorba, A. A Methodology for the Numerical Computation of Normal Forms, Centre Manifolds and First Integrals of Hamiltonian Systems. *Journal of Experimental Mathematics*, [1997].

7. Koon, W.S., Lo, M.W., Marsden, J.E., & S.D. Ross. The Genesis Trajectory and Heteroclinic Connections. *AAS/AIAA Astrodynamics Specialist Conference*,**AAS99-451**, [1999].

8. Koon, W.S., Lo, M.W., Marsden, J.E., & S.D. Ross. Heteroclinic connections between periodic orbits and resonance transitions in celestial mechanics. *Chaos*,**10**(2):427-469, [2000].

9. Marsden, J.E. Lectures on Mechanics. London Math. Soc. Lecture Note Ser., **174**, Cambridge University Press, [1992].

10. Marsden, J.E. and T.S. Ratiu. Introduction to Mechanics and Symmetry. Texts in Applied Mathematics, **17**,Springer-Verlag, [1999].

11. Marston, C.C. and N. De Leon. Reactive islands as essential mediators of unimolecular conformational isomerization: A dynamical study of 3-phospholene.*J. Chem. Phys.*, **91**:3392-3404, [1989].

12. Phillips, T., Maluendes, S., McLean, A. D., and S. Green. Anisotropic rigid rotor potential energy function for H_2O-H_2. *J. Chem. Phys.* **101**(7),[1994].

13. Uzer, T., Jaffe, C., Palacian, J., Yanguas, P., and S. Wiggins. The Geometry of Reaction Dynamics. *Nonlinearity* **15**:957-992, [2002].

14. Wiesenfeld, L., Faure, A., and T. Johann. Rotational transition states: relative equilibrium points in inelastic molecular collisions. *J. Phys. B: At. Mol. Opt. Phys.* **36**:1319-1335, [2003].