INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES
DE MONTERREY

DEPARTMENT OF CHEMISTRY

INVESTIGATION OF MILLER'S QUASICLASSICAL NUCLEAR TRAJECTORY MODEL: ANALYSIS OF SOME SIMPLE MODEL SYSTEMS.

BY
DANIEL CELIS GARZA

RESEARCH PROJECT PRESENTED AS PARTIAL REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE IN CHEMISTRY

MONTERREY, N.L. MAY 2015

INVESTIGATION OF MILLER'S QUASICLASSICAL NUCLEAR TRAJECTORY MODEL: ANALYSIS OF SOME SIMPLE MODEL SYSTEMS.

BY
DANIEL CELIS GARZA

HAS BEEN APPROVED
MAY 2015

EVALUATING COMMITTEE:

Dr. John F. Stanton, Adviser	Dr. Marcelo Videa Vargas, Adviser
Dr. Julio Leopoldo Palma Anda	Dr. Julio César Gutiérrez Vega

ACCEPTED:

Dr. Laura Eugenia Romero Robles
Head of the Bachelor's Degree in Chemistry

MSc. Luz María Gutiérrez Maldonado
Head of the Department of Chemistry

SUMMARY

Introduction

The need for new algorithms to study molecules in ever-increasing detail is not vanishing any time soon. There are algorithms for almost any problem, each with its own set of advantages and disadvantages, each suited for its specific niche in chemical, physical and pharmaceutical research. A particularly challenging one is that of molecular dynamics with electronic transitions. When a system's size is too large to conveniently treat quantum mechanically, but whose description demands knowledge of its electronic states, it lies within this inconvenient realm. Examples of such systems are large catalytic cycles, enzymatic activity, vibronic structures and other systems where trajectories and electronic states are strongly coupled to each other. These systems are often analysed using hybrid models such as semi or quasiclassical models—which are classical analogues of quantum ones. Said models scale better than their purely quantum counterparts, and are more accurate than pure molecular dynamics. Quasiclassical models don't build molecular wavefunctions like quantum methods do, but unlike molecular dynamics, they have ways of obtaining electronic information from the system. One such model is the Meyer-Miller Quasiclassical Trajectory Model explored herein.

General Characteristics of the Model and Algorithm

Many quasiclassical approaches are defined in terms of so-called 'action-angle' variables and the diabatic Meyer-Miller Hamiltonian [1] is no exception:

$$H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{n},\boldsymbol{q}) = \frac{\boldsymbol{P}^2}{2\mu} + \sum_{k=1}^{F} n_k H_{kk}(\boldsymbol{R}) + 2\sum_{k<k'=1}^{F} \sqrt{(n_k+\gamma)(n_{k'}+\gamma)} \times \cos(q_k - q_{k'}) H_{kk'}(\boldsymbol{R}) \ . \tag{1}$$

However, it is more convenient to change from action-angle variables to Cartesian variables $\{p_k, x_k\}$ via the canonical transformation:

$$x_k = \sqrt{2(n_k+\gamma)}\cos(q_k) \tag{2a}$$

$$p_k = -\sqrt{2(n_k+\gamma)}\sin(q_k) \ , \tag{2b}$$

because this approach eliminates the need to evaluate trigonometric functions during numerical integration, and makes the derivation of the equations of motion a less dolorous affair.

The initial conditions are set within the action-angle frame, using Monte-Carlo (MC) sampling methods and the formulae:

$$n_k(0) = N_k + \gamma(2 \cdot RN_k - 1) \tag{3a}$$
$$q_k(0) = 2\pi \cdot RN'_k \ , \tag{3b}$$

where $N_k = 0$ means the k^{th} state is unoccupied, and $N_k = 1$ means the k^{th} state is occupied, γ is the zero-point energy contribution parameter, and finally RN_k and RN'_k are two uniformly distributed random numbers $\in [0, 1]$ interval.

After all trajectories have been integrated, eqs. (2a) and (2b) can be used to calculate the final values of $n_k(t)$,

$$n_k = \frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \gamma \ , \tag{4}$$

(the angle variable is irrelevant).

A distinguishing feature of the MM model explored here is the symmetrical windowing of the electronic states in terms of the Heaviside function, $h(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$:

$$W_{N_k}(n_k) = \frac{1}{\Delta n} h\left(\frac{\Delta n}{2} - |n_k - N_k|\right) \ . \tag{5}$$

In order for $W_{N_k} \neq 0$, $n_k \in [N_k - \Delta n/2,\ N_k + \Delta n/2]$. Thus making Δn the function's width—which in this case is defined as, $\Delta n = 2\gamma$. It is worth noting that the window function for a state k is the product of such functions for each electronic degree of freedom. In the case of the sample problems, there are two electronic states ($F = 2$). For the first one, $(N_1, N_2) = (1, 0)$, the window function is, $W_1(n_1) \cdot W_0(n_2)$; similarly for the second state, $(N_1, N_2) = (0, 1)$ we have, $W_0(n_1) \cdot W_1(n_2)$.

A trajectory is a assigned a final state k if the final values of n_k comply with the following inequality:

$$N_k - \gamma \leq n_k \leq N_k + \gamma \ , \tag{6}$$

in terms of Cartesian coordinates this means:

$$N_k \leq \frac{1}{2}x_k^2 + \frac{1}{2}p_k^2 \leq N_k + 2\gamma \ . \tag{7}$$

It's worth noting that the criteria must be simultaneously met for all k electronic degrees of freedom—for when they are not, we get a 'mixture' of states, which is incompatible with the model, and the calculation has to be restarted until the initial conditions allow them to be met.

In certain problems, care must also be taken to check whether the particle has been transmitted i.e. has left the integration area *opposite* to its initial side, or reflected

i.e. has left integration area from the *same* side to where it started. It is of the utmost importance to take this into account when tackling problems where particle trajectories, as well as transition probabilities interest us.

The final window functions are then calculated via eqs. (5) and (7), and Monte-Carlo (MC) averaged. Lastly, the transition probabilities from initial to final state, $P_{f\leftarrow i}$, are calculated by multiplying the initial and final window functions (the *and* rule of probability), and dividing by the sum of the corresponding quantities for all possible final states,

$$P_{f\leftarrow i} = \frac{\left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle}{\sum_{f=1}^{F} \left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle} \ . \tag{8}$$

As previously stated—in cases where reflection and transmission are also important—eq. (8) changes to:

$$P^{a}_{f\leftarrow i} = \frac{\left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle_a}{\sum_{a=r,\ t} \left(\sum_{f=1}^{F} \left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle \right)_a} \ , \tag{9}$$

where $\delta_{ij} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$, $\langle \cdots \rangle$ denotes Monte-Carlo averaging, $r \equiv$ reflection and $t \equiv$ transmission. The implementation of both procedures does not differ a lot, the major difference is that in the case of eq. (8), the numerator can be implemented as a single 1D array with F items; in the case of eq. (9) it can be implemented as two (one for reflection and one for transmission) 1D arrays with F items (or a 2D array with $2 \times F$ items).

All trajectories were integrated using the standard RK4-Gill method.

Diabatic and Adiabatic Hamiltonians

As previously stated, it is easier to work with the MM Hamiltonian (eq. (1)) in Cartesian variables:

$$\begin{aligned} H(\boldsymbol{P}, \boldsymbol{R}, \boldsymbol{n}, \boldsymbol{q}) = \frac{\boldsymbol{P}^2}{2\mu} + \sum_{k=1}^{F} \left(\frac{1}{2} p_k^2 + \frac{1}{2} x_k^2 - \gamma \right) H_{kk}(\boldsymbol{R}) \\ + \sum_{k<k'=1}^{F} (p_k p_{k'} + x_k x_{k'}) H_{kk'}(\boldsymbol{R}) \ . \end{aligned} \tag{10}$$

By utilising eqs. (2a), (2b) and (4); the angle-difference identity $\cos(\alpha - \beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)$; the definition of the arithmetic mean PES,

$$\bar{H} = \frac{1}{F}\sum_{k=1}^{F} n_k H_{kk} \ ; \tag{11}$$

and the normalisation condition,

$$1 = \sum_{k=1}^{F} n_k \ ; \tag{12}$$

we can re-write [1] the Hamiltonian in the following way[1]:

$$\begin{aligned} H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{p},\boldsymbol{x}) &= \frac{\boldsymbol{P}^2}{2\mu} + \bar{H}(\boldsymbol{R}) \\ &+ \sum_{k<k'=1}^{F} \left\{ \begin{array}{c} \frac{1}{F}(H_{kk}(\boldsymbol{R}) - H_{k'k'}(\boldsymbol{R})) \cdot \left(\frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \frac{1}{2}p_{k'}^2 - \frac{1}{2}x_{k'}^2\right) \\ + H_{kk'}(\boldsymbol{R}) \cdot (p_k p_{k'} + x_k x_{k'}) \end{array} \right\} \ . \end{aligned} \tag{13}$$

The adiabatic Hamiltonian is treated in exactly the same way, but being a diagonalised version of the adiabatic Hamiltonian, it lacks an explicit non-diagonal term, uses the eigenvalues of the adiabatic PES (E_k), and contains a so-called 'mixing angle' term, $\omega(\boldsymbol{R})$, in the nuclear kinetic energy term:

$$\begin{aligned} H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{p},\boldsymbol{x}) &= \frac{|\boldsymbol{P} + \nabla\boldsymbol{P}|^2}{2\mu} + \bar{E}(\boldsymbol{R}) \\ &+ \sum_{k<k'=1}^{F} \frac{1}{F}(E_k(\boldsymbol{R}) - E_{k'}(\boldsymbol{R})) \cdot \left(\frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \frac{1}{2}p_{k'}^2 - \frac{1}{2}x_{k'}^2\right) \ , \end{aligned} \tag{14}$$

where,

$$\nabla\boldsymbol{P} = \sum_{k<k'=1}^{F} \omega(\boldsymbol{R}) \cdot (p_k x_{k'} - p_{k'} x_k) \tag{15a}$$

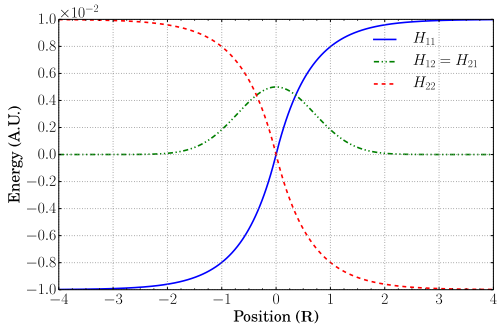
$$\Delta\boldsymbol{P} = \sum_{k<k'=1}^{F} \frac{\partial\omega(\boldsymbol{R})}{\partial\boldsymbol{R}} \cdot (p_k x_{k'} - p_{k'} x_k) \tag{15b}$$

$$\omega(\boldsymbol{R}) = \frac{1}{2}\arctan\left(\frac{2H_{kk'}(\boldsymbol{R})}{H_{kk}(\boldsymbol{R}) - H_{k'k'}(\boldsymbol{R})}\right) \ . \tag{15c}$$

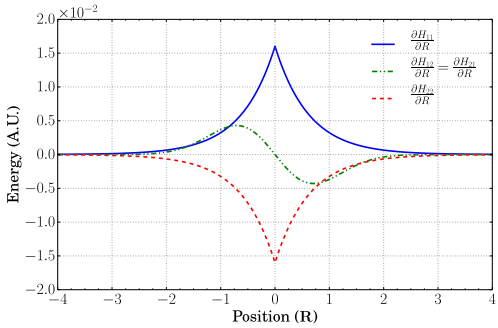
Model Systems

All problems analysed in this work have only two electronic degrees of freedom (electronic states), $\therefore F = 2$. Furthermore, all problems assume $\mu = m_k = 2000$, and all units are in atomic units.

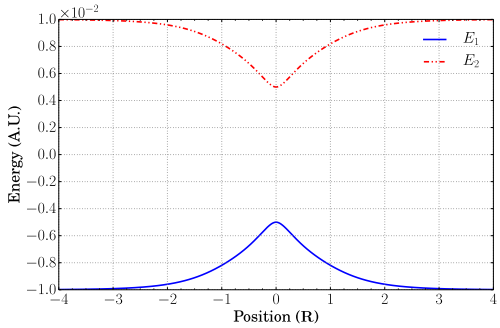
[1] For details about the transformation see appendix B.1.



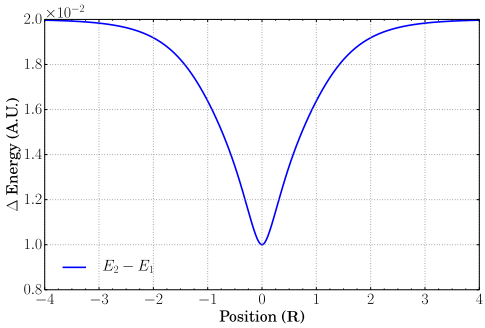
(a) Diabatic PES.



(b) Diabatic PES derivatives.



(c) Adiabatic PES.



(d) ΔE between adiabatic PES.

Figure 1: Single avoided crossing PES.

Tully analysed three avoided crossing problems with his least-hops algorithm [2]. These three problems, along with two others which use the spin-boson model for condensed-phase dynamics were analysed here—all of which were analysed for initial states 1 and 2. It is worth noting that the smaller the difference between two adiabatic PES becomes, the likelier an electronic transition from one to the other becomes.

Single Avoided Crossing

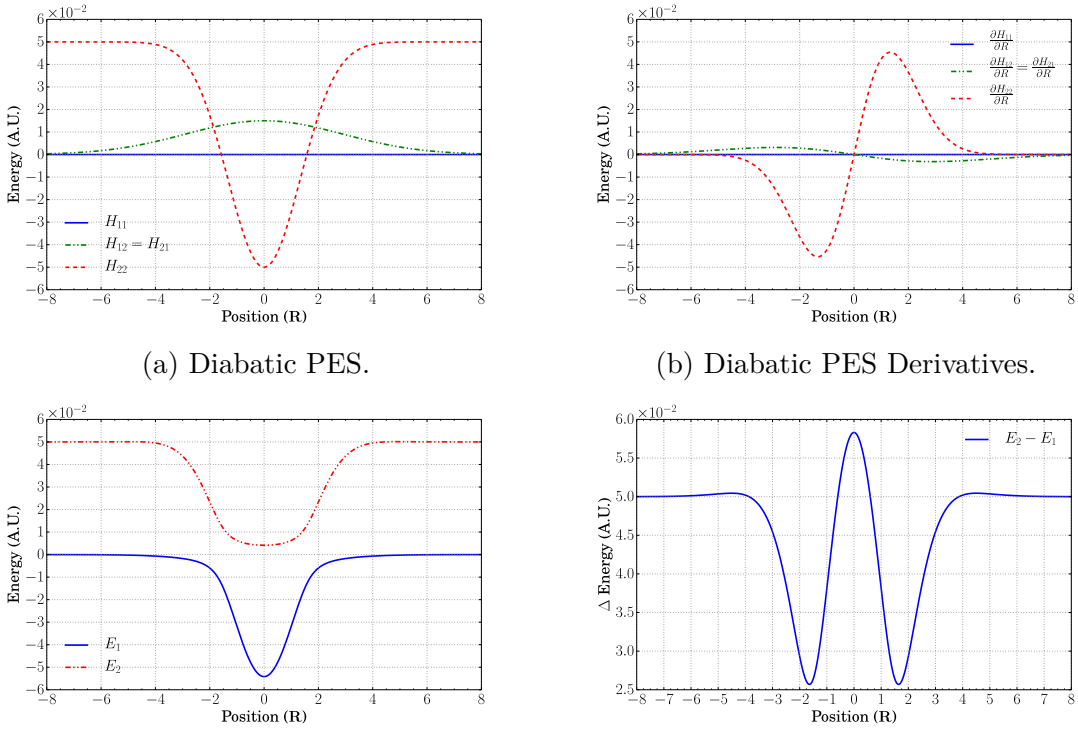
The diabatic potential energy surfaces (PES) for the single avoided crossing were defined by Tully [2] as:

$$H_{11}(R) = \begin{cases} A\left(1 - e^{-BR}\right) & R \geq 0 \\ -A\left(1 - e^{BR}\right) & R < 0 \end{cases} \tag{16a}$$

$$H_{22}(R) = -H_{11}(R) \tag{16b}$$

$$H_{12}(R) = H_{21}(R) = Ce^{-DR^2}\ , \tag{16c}$$

where $A = 0.01$, $B = 1.6$, $C = 0.005$, and $D = 1$. The diabatic PES, their derivatives, adiabatic PES and ΔE between adiabatic PES are plotted in fig. 1.



(a) Diabatic PES.

(b) Diabatic PES Derivatives.

(c) Adiabatic PES.

(d) ΔE between adiabatic PES.

Figure 2: Double avoided crossing PES.

Double Avoided Crossing

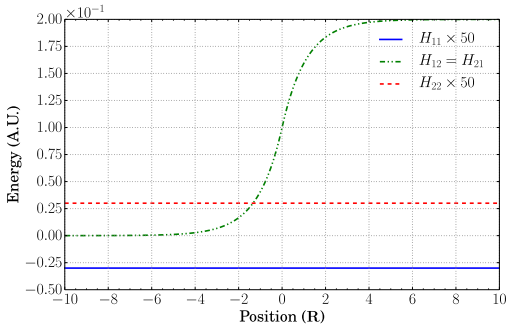
The diabatic potential energy surfaces (PES) for the double avoided crossing were defined by Tully [2] as:

$$H_{11}(R) = 0 \tag{17a}$$

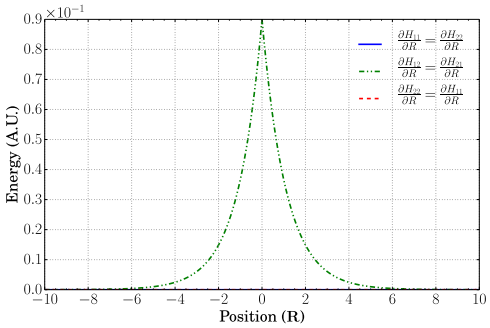
$$H_{22}(R) = -Ae^{-BR^2} + E_0 \tag{17b}$$

$$H_{12}(R) = H_{21}(R) = Ce^{-DR^2} \ , \tag{17c}$$

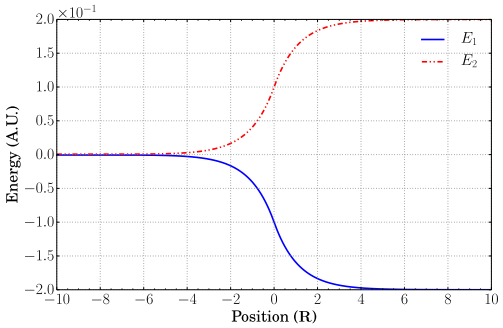
where $A = 0.1$, $B = 0.28$, $C = 0.015$, $D = 0.06$, y $E_0 = 0.05$. The diabatic PES, their derivatives, adiabatic PES and ΔE between adiabatic PES are plotted in fig. 2.



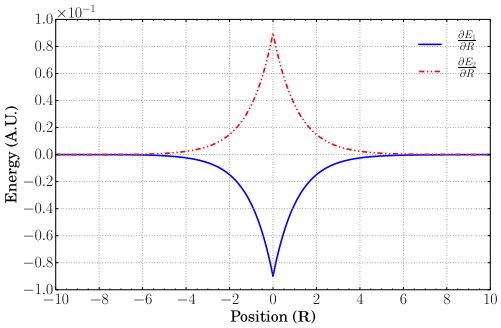
(a) Diabatic PES.



(b) Diabatic PES derivatives.



(c) Adiabatic PES.



(d) Adiabatic PES derivatives.

Figure 3: Extended coupling PES.

Extended Coupling

The diabatic potential energy surfaces (PES) for the extended coupling were defined by Tully [2] as:

$$H_{11}(R) = -A \tag{18a}$$

$$H_{22}(R) = -H_{11} \tag{18b}$$

$$H_{12}(R) = H_{21}(R) = \begin{cases} B(2 - e^{-CR}) & R \geq 0 \\ Be^{CR} & R < 0 \end{cases} \tag{18c}$$

where $A = 6 \times 10^{-4}$, $B = 0.1$, and $C = 0.9$. This problem has non-vanishing non-diagonal elements in the Hamiltonian matrix. Said elements are perturbations from a pure state, and when they do not vanish when $R \to \pm\infty$, one must use the adiabatic Hamiltonian. Figure 3 shows the diabatic and adiabatic PES and their derivatives (the difference between both adiabatic PES is not shown because it's fairly obvious).

Spin-Boson Model for Condensed-Phase Dynamics

This problem is vastly different from the others. In our case, the model describes a 1D system of M coupled oscillators. More specifically, a one-dimensional lattice made

up of M oscillating nuclei which posses bulk electronic states. What is measured here are not trajectories, but rather the time-dependent difference in the probability of finding the system in either of two electronic states; when the initial state = 1, $D(t) = P_{1\leftarrow 1} - P_{2\leftarrow 1}$, when the initial state = 2, $D(t) = P_{1\leftarrow 2} - P_{2\leftarrow 2}$.

The diabatic PES are defined as follows:

$$H_{11}(\boldsymbol{Q}) = V_0(\boldsymbol{Q}) + V_1(\boldsymbol{Q}) + \epsilon \tag{19a}$$
$$H_{22}(\boldsymbol{Q}) = V_0(\boldsymbol{Q}) - V_1(\boldsymbol{Q}) - \epsilon \tag{19b}$$
$$H_{12}(\boldsymbol{Q}) = H_{12}(\boldsymbol{Q}) = \Delta \ , \tag{19c}$$

where,

$$V_0(\boldsymbol{Q}) = \sum_{k=1}^{M} \frac{1}{2} m_k \omega_k^2 Q_k^2 \tag{20a}$$
$$V_1(\boldsymbol{Q}) = \sum_{k=1}^{M} c_k Q_k \ . \tag{20b}$$

For our purposes, the frequencies, ω_k, are uniformly distributed $\in [0.01\omega_c,\ 4\omega_c]$ [3], where ω_c is known as the *characteristic frequency*, and defines the system's overall time scale. The observant reader will realise this does not make a lot of sense on its own, because not all frequencies contribute to a system's energy in equal amounts. Which is why the coupling parameters, c_k, are chosen so they obey the distribution:

$$J(\omega) = \frac{\pi}{2} \sum_{k=1}^{M} \frac{c_k^2}{m_k \omega_k} \delta(\omega - \omega_k) \ , \tag{21}$$

where $J(\omega)$ is an Ohmic distribution defined as:

$$J(\omega) = \frac{\pi}{2} \alpha \omega \exp\left(-\frac{\omega}{\omega_c}\right) \ , \tag{22}$$

where α is the Kondo parameter (a coupling strength parameter). After equating eqs. (21) and (22), and integrating with respect to ω, we find the values of c_k to be:

$$c_k = \sqrt{\frac{2}{\pi} \Delta\omega m_k \omega_k J(\omega_k)} = \omega_k \sqrt{\alpha \Delta\omega m_k \exp\left(-\frac{\omega_k}{\omega_c}\right)} \tag{23a}$$
$$\Delta\omega = \frac{\omega_{max} - \omega_{min}}{M - 1} \ . \tag{23b}$$

The initial electronic conditions are initiated in the same manner as before. However, the nuclear ones are not. As stated before, the model describes oscillating nuclei in a 1D lattice, and their momenta and positions are assumed to follow the Wigner

distribution:

$$\rho(\boldsymbol{P},\boldsymbol{Q}) = \prod_{k=1}^{M} \exp\left(-a_k \frac{P_k^2}{2m_k}\right) \cdot \exp\left(-a_k \frac{1}{2} m_k \omega_k^2 \left[Q_k + \frac{c_k}{m_k \omega_k^2}\right]^2\right) \tag{24a}$$

$$a_k = \frac{2}{\omega_k} \tanh\left(\frac{\beta\omega_k}{2}\right) \ . \tag{24b}$$

Which, in reality, is a bivariate normal distribution. Fortunately, the lack of a covariant term implies that positions and momenta can be independently sampled from normally distributed random numbers. In other words, the nuclei follow independent normal distributions for their momenta and positions. It is also worth noting that the argument for the total distribution is basically a scaled quasiclassical analogue of the quantum harmonic oscillator Hamiltonian operator ($\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2} m\omega^2 \hat{x}^2$), which makes a lot of intuitive sense, because it means the nuclear parameters are sampled according to the oscillators' normally distributed kinetic energy.

Sampling is done in the traditional way: assuming X is a normally distributed random number with unitary standard deviation and mean equal to zero, a normally distributed random number G with standard deviation equal to σ and mean equal to μ can be calculated by $G = \sigma X + \mu$. Given the definition of the normal distribution,

$$G(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{25}$$

the standard deviations and mean values of the nuclei's momenta and positions are:

$$\sigma_{P_k} = \sqrt{\frac{m_k \omega_k}{2\tanh\left(\frac{\beta\omega_k}{2}\right)}} \tag{26a}$$

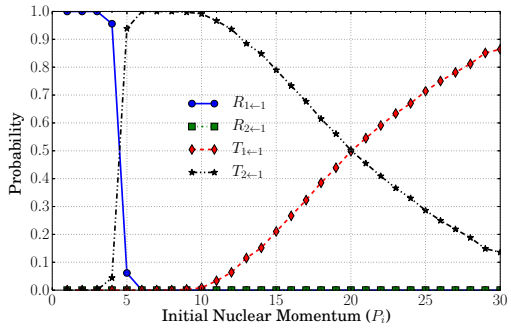
$$\mu_{P_k} = 0 \tag{26b}$$

$$\sigma_{Q_k} = \sqrt{\frac{1}{2m_k\omega_k \tanh\left(\frac{\beta\omega_k}{2}\right)}} \tag{26c}$$

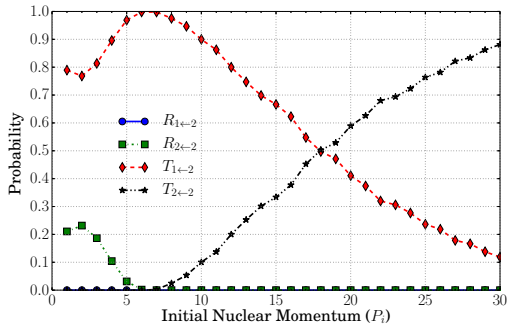
$$\mu_{Q_k} = -\frac{c_k}{m_k \omega_k^2} \ . \tag{26d}$$

Results

The code written to implement the model was found to scale linearly with integration step, number of MC repetitions and number of cores used. It was also found that accuracy is more dependent on the number of MC repetitions than the size of the integration step, h. In fact, the integration step could be made relatively large (how large depends on the problem) without significantly affecting accuracy, but drastically reducing computational time. There was also no significant difference between using $\gamma = \frac{\sqrt{3}-1}{2}$ and $\gamma = 0.366$ (the actual value is very similar, but there is no need to



(a) Transition probabilities for Reflection (R) and Transmission (T) for $h = (0.012P_i)^{-1}$ and initial state = 1.



(b) Transition probabilities for Reflection (R) and Transmission (T) for $h = (0.012P_i)^{-1}$ and initial state = 2.

Figure 4: Single avoided crossing transition probabilities.

evaluate square roots and divisions all the time if we use the approximation); [1] gives the theoretical justification for this value, but it can be adjusted on a case by case basis. Accuracy was also not noticeably affected by parallel runs, presumably due to the random number generator used—which is thread safe—but not optimal for parallelisation, because it generates numbers in sequence from a single source rather than in parallel[2] (it only affects execution speed).

The number of MC repetitions (MC reps) = 15000 and the value of $\gamma = 0.366$ unless stated otherwise. Reflections are denoted as R and transmissions as T, electronic transitions from an initial state, i, to a final state, f, are denoted as $f \leftarrow i$.

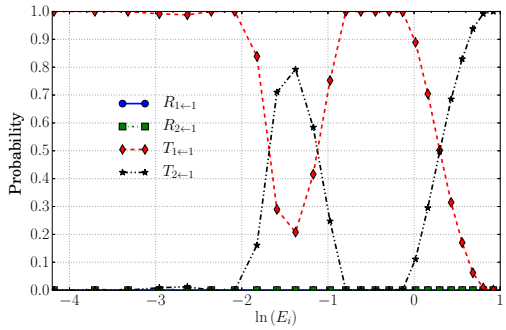
Single Avoided Crossing

The results for when initial state = 1 are very similar—though not exactly the same—as those found in [1]. However, this is most likely down to the fact that they used 50000–100000 trajectories as well as there being a strong possibility that the compiler, RNG, and integration step differ. Figure 4 showcases the results for initial state = 1, 2.

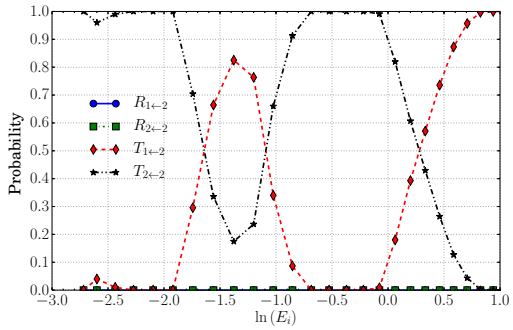
Despite this being the simplest problem tackled herein, it presents some fairly interesting behaviours.

First, we shall tackle fig. 4a. The behaviour of $R_{1\leftarrow1}$—whose probability is seen to rapidly decline as nuclear momentum increases—can be explained by referring back to fig. 1. For small values of nuclear momentum, there is not enough kinetic energy to scale over the potential barrier (akin to an 'activation energy') and continue in the original direction, thus the particle is reflected back. The behaviour of $R_{2\leftarrow1}$ is explained by the same reason; once there is enough kinetic energy to reach a point where a transition is likely to happen, there would also be enough energy to continue

[2]See https://gcc.gnu.org/onlinedocs/gfortran/RANDOM_005fNUMBER.html



(a) Transition probabilities for Reflection (R) and Transmission (T) for $h = (0.0125P_i)^{-1}$ and initial state = 1.



(b) Transition probabilities for Reflection (R) and Transmission (T) for $h = (0.0125P_i)^{-1}$ and initial state = 2.

Figure 5: Double avoided crossing transition probabilities.

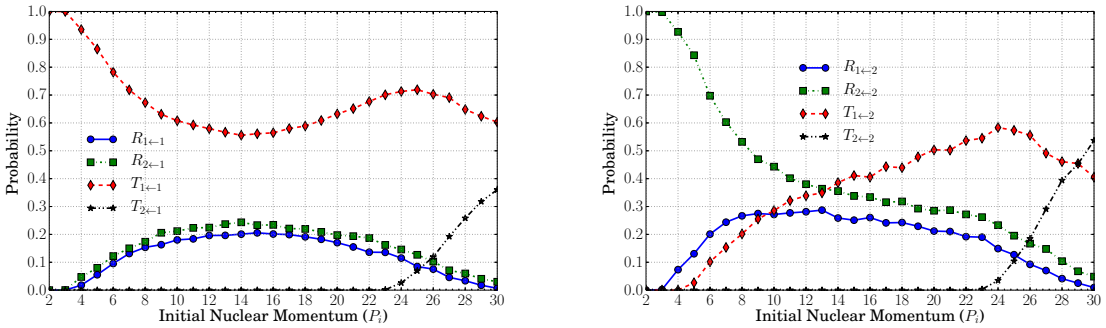
on to the other side. Thus $R_{2\leftarrow1}$ remains relatively close to zero throughout. The others can be explained by the fact that transition probabilities are not only a function of the distance between adiabatic PES, but also on the time it takes the particle to move away from the areas where this value is small. This explains why the $T_{2\leftarrow1}$ and $T_{1\leftarrow1}$ transitions behave the way they do. In the case of $T_{2\leftarrow1}$ lower nuclear momenta allow more time for an electronic transition to occur; however, after a certain value of nuclear momentum, it begins to decrease in favour of $T_{1\leftarrow1}$ because time the particle travels so fast that it cannot interact long enough to undergo an electronic transition.

For fig. 4b the behaviour is more nuanced, but down to mostly the same reasons. The $R_{1\leftarrow2}$ transition remains effectively zero throughout because at such low values of nuclear momenta two things can happen: the particle can't get past the dip in E_2 (if it doesn't move to the lower energy state) because it gets reflected back by the off-diagonal PES—corresponding to the $R_{2\leftarrow2}$ transition—or it moves down to the lower energy state, where the surplus potential energy is converted into kinetic energy (conservation of energy) and a $T_{1\leftarrow2}$ transition is observed. For higher values of nuclear momentum, the particle has enough energy to be transmitted, but there is not enough time for the particle to linger long enough for there to be a state transition, and $T_{2\leftarrow2}$ is observed instead, and the higher the nuclear momentum the less time there is and the likelier said process becomes.

Double Avoided Crossing

Again the results for when initial state = 1 are very similar—but not exactly the same—as those found in [1]. Which can be attributed to the same factors as before. Figure 5 shows the results for initial state = 1, 2.

Once more, we shall first analyse the behaviour for initial state = 1, shown in fig. 5a. The lack of $R_{1\leftarrow1}$ and $R_{2\leftarrow1}$ transitions is easily explained by the fact there is



(a) Transition probabilities for Reflection (R) and Transmission (T) for $h = (0.10125P_i)^{-1}$, 30000 MC reps, and initial state = 1.

(b) Transition probabilities for Reflection (R) and Transmission (T) for $h = (0.10125P_i)^{-1}$ and initial state = 2.

Figure 6: Extend coupling transmission probabilities.

no energy barrier in either the diagonal diabatic or adiabatic PES (figs. 2a and 2c), therefore the particle only requires a small amount of momentum for the particle to be transmitted. In fact, in low momentum tests, the particle was accelerated by the adiabatic energy wells (potential energy was transferred into kinetic energy). The more interesting features are the Stückelberg oscillations for both $T_{1\leftarrow1}$ and $T_{2\leftarrow1}$, which are often attributed to quantum interference effects. In order to comprehend these one must only turn to fig. 2d, which shows the difference between adiabatic PES. One can see that $E_2 - E_1$ varies quickly and by relatively large amounts, causing the system's electronic coordinates to vary wildly and quickly. For small momenta, the particle generally does not have enough energy to switch electronic states, leading to the relatively flat appearance of both $T_{1\leftarrow1}$ and $T_{2\leftarrow1}$ from -4 to -2. However, as momentum increases, there is enough energy to make one jump but not a lot to make the second one, leading to the behaviour seen from -2 to -1. As momentum keeps increasing, there is enough energy for the second jump, and we get the behaviour we see from -1 to 0. Finally, when momentum is very large, the particle carries out one transition but does not linger long enough to make a second, leading to the behaviour we see from 0 to 1.

When the initial state = 2 (fig. 5b), the behaviour is very similar but inverted to the one described in the previous paragraph. In other words, the transmission which preserves the initial state, $T_{2\leftarrow2}$ in fig. 5b, is very similar in shape to the transmission which preserves the initial state, $T_{1\leftarrow1}$ in fig. 5a. And the transmission which does not preserve the initial state, $T_{1\leftarrow2}$ in fig. 5b, is also very qualitatively similar to the transition which does not preserve the initial state, $T_{2\leftarrow1}$ in fig. 5a. Which means they have similar explanations, with only the exact points at which these 'domains' appear, varying between both initial states.

Extended Coupling

Once more, the results for when initial state = 1 are very similar—but not exactly the same—as those found in [1]. Which can again be attributed to the same factors as before. Figure 6 shows the results for initial states = $1, 2$.

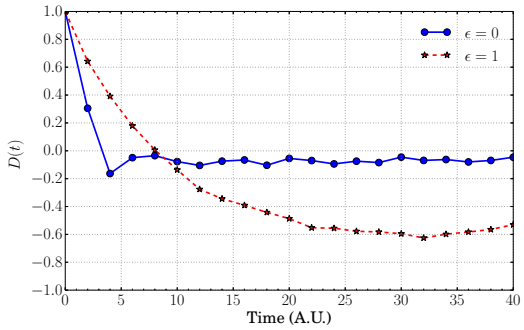
Following previously set precedents, we shall start the analysis with fig. 6a for which initial state = 1. The behaviour of $T_{1\leftarrow 1}$ at low momenta is due to the fact that E_1 is highly attractive, so the particle is easily transmitted without strongly interacting with any other PES. However, as the momentum increases the particle starts interacting more strongly with the non-diagonal elements of the hamiltonian matrix, H_{12}, and it becomes more likely to either be reflected by it with or without an electronic transition, giving rise to the bump in probability of $R_{1\leftarrow 1}$ and $R_{2\leftarrow 1}$, and the dip in the probability of $T_{1\leftarrow 1}$. It is only at higher nuclear momenta that it is possible for the particle to experience an electronic transition and have enough momentum to work against the repulsive interaction with and E_2, leading to a probability increase of $T_{2\leftarrow 1}$.

For the case when the initial state = 2 (fig. 6b), the system's behaviour changes drastically. At low nuclear momenta, the particle does not have the required energy to be transmitted or undergo an electronic transition, leading to the shape of $R_{2\leftarrow 2}$. However, as the momentum increases, the particle starts being able to change electronic state and be transmitted, leading to the increase in the probabilities of $R_{1\leftarrow 2}$ and $T_{1\leftarrow 2}$. As the momentum is increased further, the probability of having enough energy to transition into the lower electronic state and be transmitted increases, leading to the drop in $R_{2\leftarrow 2}$ and $R_{1\leftarrow 2}$, and increase of $T_{1\leftarrow 2}$. As the nuclear momentum increases further, then the particle is travelling fast enough that it can overcome the repulsive effect of E_2 and interact relatively little with E_1, leading to the increased probability of observing $T_{2\leftarrow 2}$.

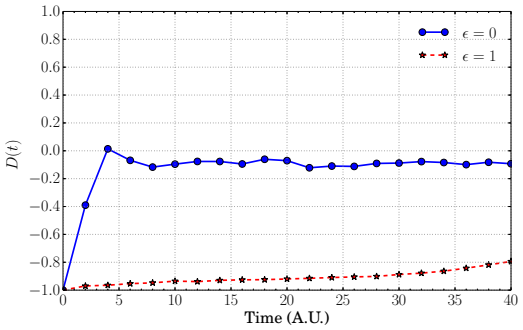
Spin-Boson Model for Condensed-Phase Dynamics

Some details were left out of [1] in regards to this problem—namely the value of ω_c, and the range and distribution of ω_k—so it was assumed that $\omega_c = 1$ and ω_k uniformly distributed $\in [0.01\omega_c,\ 4\omega_c]$ [3], and everything else was defined from there; which means there is no standard with which to compare results. Figures 7 and 8 show the results of four systems (two symmetric and two asymmetric) characterised by different parameters, for initial electronic states = $1, 2$. The calculations were carried out for 100 nuclei ($M = 100$).

Figure 7 shows incoherent (non-oscillatory) relaxation dynamics for all four systems. It is particularly interesting how much the behaviour changes between the symmetric and asymmetric version of each problem. In both cases, the asymmetric system favours the second state. Which means that the second state has a lower energy than the first. This notion is also evidenced by the fact that both symmetric



(a) Initial state = 1. For both systems $h = 0.01$.



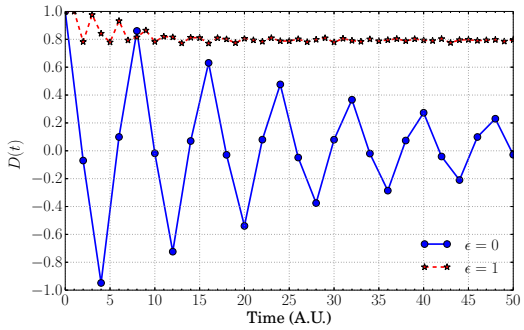
(b) Initial state = 2. For the symmetric problem, $h = 0.1$ and MC reps = 5000. For the asymmetric problem, $h = 0.01$ and MC reps = 30000.

Figure 7: Spin-boson calculations for symmetric ($\epsilon = 0$) and asymmetric ($\epsilon = 1$) systems. $\alpha = 0.09,\ \beta = 0.25,\ \Delta = (2.5)^{-1}$.

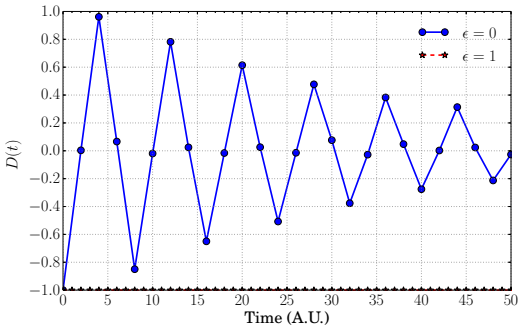
systems stabilise at negative values of $D(t)$, signifying that the probability of finding the system in state 2 is higher than that of finding it in state 1. This can potentially be observed from eq. (19).

On the other hand, fig. 8 shows both, coherent (oscillatory) and incoherent (non-oscillatory) relaxation dynamics. For both initial states, the symmetric system manifests highly oscillatory behaviours. But the surprising results arise from their asymmetric counterparts. Such results mean that for our chosen parameters, the dynamical system described by the equations of motion has large regions of stability in the electronic phase space. Such phenomena are not unheard of—where the same equations with different parameters yield vastly different results, ranging from nicely behaved to chaotic systems. Suffice to say, this was completely unexpected (and probably highly coincidental).

To conclude with this discussion, it must be said that the disparity in integration step and MC reps is down to each individual system's run time. The aforementioned quantities were adjusted so that no run exceeded 17 hours. There was some over-compensation in a few systems due to the highly variable nature of their run times, which in one instance was observed to exceed 250000%, though less extreme examples fell in the range of 300% to 1000%. This phenomenon is attributed to the large number of uniformly and normally distributed random numbers required to set the initial conditions—given that some of them do not allow the selection criteria to be met. The problem was compounded by the random seed allocation at every MC rep, and by the fact that the selection criteria (eq. (7)) must be met at every plot point, otherwise the MC rep has to be restarted.



(a) Initial state = 1. For the symmetric problem, $h = 0.1$ and MC reps = 5000. For the asymmetric problem, $h = 0.05$ and MC reps = 30000.



(b) Initial state = 2. For the symmetric problem, $h = 0.1$ and MC reps = 5000. For the asymmetric problem, $h = 0.05$ and MC reps = 30000.

Figure 8: Spin-boson calculations for symmetric ($\epsilon = 0$) and asymmetric ($\epsilon = 1$) systems. $\alpha = 0.1,\ \beta = 12.5,\ \Delta = (2.5)^{-1}$.

Conclusions

The most recent version of the MM model was successfully implemented in FORTRAN 2008. The code was validated with four simple model systems. New results for all four systems were obtained. Future work should be carried out to modify the code so it can read input files and be applied to arbitrary systems. The use of an adaptive integration algorithm may also be worth investigating, as it can make calculations faster and more accurate. Lastly, attempts must be made to eliminate the need for analytic PES so the model may be applied to real systems.

ACKNOWLEDGEMENTS

First, I would like to thank my academic mentors—who I have come to consider my friends. They have been many, their origins and locations span the globe and I cannot name them all, but there are a few who I would like to mention—as their influence on me, I consider the greatest. Starting with Prof. John F. Stanton, who after only two e-mails, told me he would like to be my adviser. If I had been in his position, I would've never risked it on so little information, but I'm glad he did… because it was awesome. My academic adviser, Prof. Marcelo Videa Vargas, for allowing this to happen, and for all the advice and teachings over the years. I'd also like to thank Prof. Anatoly B. Kolomeisky and soon-to-be PhD. Hamid Teimouri for having accepted me as part of the team, as well as for all the help, guidance, advice and philosophical conversation during my 2014 summer internship at the Kolomeisky Group. There's also Prof. Víctor M. Jiménez Pérez, PhD. Concepción "Conny" García, PhD. Rodrigo Chan Navarro and Prof. Víctor M. Rosas García, who also welcomed me as part of the team, as well as helped, guided, and advised me during my 2013 summer internship at the Jiménez-Pérez Group. I'd also like to thank Prof. Bernard Micheli Masson, whose classes kept me studying chemistry. Mr. Casserly, whose secondary school chemistry class made me study chemistry, and for helping me develop a critical and objective mind; Mr. Walker, my secondary school ICT teacher, whose words of praise, encouragement and lament over my return to Mexico, I still turn to in times of need and desperation; Mrs. Burrow, Mrs. Milner and Miss Clarke, whose world-views and philosophies I have integrated into my own; and lastly, Mrs. Wade, for all her help when I arrived in a strange, grey land, and upon leaving a grey home.

I also wish to thank www.reddit.com, www.stackexchange.com and www.wikipedia.org, whose communities freely provided me with tools, knowledge and support I could only have otherwise acquired by sinking more money and time into overpriced books, articles and further schooling.

Next in line are my friends—who I have also come to consider my mentors. Namely, I would like to thank *The Spectral League*, *The Disciples*, PhD. Julio L. Palma Anda, and my Senseis Francisco "Akira" Espinoza Mancera, Kieth Prole and Sacha Martin-Luther King. The reasons for their appearance here are as varied as they are, but the one thing they all have in common, is that they have bestowed upon me the tools and responsibility to continually improve myself.

And last but certainly not least, my family. And as I have mentioned before, my definition of family is not the traditional definition concerning an arbitrarily defined amount of shared genetic material; but rather that which is given in Disney's *Lilo and Stitch*: "Ohana means family. Family means nobody gets left behind, or forgotten." It is to these people, who have never left my side—or my mind—that I owe the greatest, most heartfelt of thanks. I would name them all, but there is no need, for they instinctively know who they are. I thank you all for making me the person I am… I hope you've made yourselves proud.

DEDICATION

To my Ohana.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Computational Methods in Chemistry

Computantional chemistry enables the study of chemical systems in ways that traditional techinques cannot [4]. For example, solvent effects in chemical reactions [5] without carrying out the reaction under different solvents and isolating intermediaries. Computational chemistry can also be very valuable for the pharmaceutical industry [6]; particularly the area of cheminformatics, which can greatly speed up the search and sorting of potentially bioactive molecules [7, 8]. For example, the discovery DNA gyrase inhibitors with higher affinity than previously known ones [9–11]. These techniques are usually extremely fast, for instance, in 2001 a laptop with a 3 GHz processor estimated the ADME [12] physical properties of 71,500 compounds in 3 minutes using the QikProp package [13], providing relevant information for further study.

Nowadays, computational chemistry has permeated popular culture in the form of Folding@Home and the Clean Energy Project. Folding@Home currently outputs $48,000$ teraflops of computing power ($48,000 \times 10^{12}$ floating point operations per second). F@H focuses on the study of misfolded proteins in Alzheimer's, Huntington's, ALS, Mad Cow, Parkinson's, many cancers, among others [14]. On the other hand, the Clean Energy Project focuses on the search and analysis of possible candidates for use in renewable energy projects, and had a cumulative total of 22,564 CPU years up to Jan 17 2014 [15].

However, behind the broad range of applications, lie the theory and algorithms which make it all possible. Utilising theoretical techniques and efficient algorithms one can calculate, observe, and explore some of the most fundamental characteristics of chemical systems. Namely: vibrational modes, intermolecular interactions, and electronic structure, all of which can be analysed independently and on different levels of detail [16]; though sometimes, the problem requires some degree of interdependency.

Computational methods typically fall into four groups [17, 18]: 1) *Ab initio* meth-

Table 1.1: Scaling differences in the calculation* of Coulombic repulsion in two DFT methods (times in minutes). The calculations were carried out on alanine oligomers of 5, 10 and 15 amino acids. Scaling as a function of basis set went from $O(N^4)$ to $O(N^2)$ with the new method. Table edited from [27].

Basis Set†	5(old)	5(new)	10(old)	10(new)	15(old)	15(new)
6-31G(df,pd)	39	27	145	98	263	169
6-31G+(df,pd)	109	46	736	160	1240	362
cc-pvdz	56	20	177	79	369	143
cc-pvtz	361	85	1165	280	2482	551

∗ Calculations carried out on a single 2 GHz processor of an Opteron cluster.

† Basis sets in the form of $X - YZ$G mean that internal orbitals are composed of X Gaussian primitives (G), while valence orbitals are composed of two functions each; the first is composed of a linear combination of Y G; the second of a linear combination of Z G. The + sign indicates the use of diffuse functions, and the parentheses the use of polarisation functions composed of a linear combination of orbital type p, d, f functions as the case may be. Basis sets of the type cc-pv$n$z where n = d, t, q, 5, 6... where (d = double, t = triple, q = quadruple) contain correlation-consistent polarisation functions for valence orbitals; cc-p means 'correlation-consistent polarised' and 'v' indicates that they only model valence orbitals [16, 28, 29].

ods start from physical constants and are purely quantum-mechanical, dealing with all the problems N-body interacting systems bring. 2) Molecular Dynamics (MD) methods apply classical mechanics to chemical systems. They are typically used in the analysis of large-scale systems, where other methods would take too long for a small increase in overall accuracy. 3) Semi-empirical methods use experimental parameters as well as *ab initio* techniques, sacrificing accuracy in the name of speed. 4) Density Functional Theory (DFT) methods are often mentioned as a subset of *ab initio* methods. DFT methods create a spatially dependent electron density function, reducing the problem from N electrons with 3N spacial dimensions to 3 spatial dimensions. Thus greatly reducing computational time. The caveat is that DFT methods are not suitable for strongly correlated systems. They can be extended to accomodate time dependency, such methods are called TD-DFT.

Which method is best suited for one's needs depends on the complexity of the system to be studied, the desired accuracy, and available computational resources [19]. Purely quantum mechanical methods such as *ab initio* and DFT are particularly afflicted by scaling[1] issues [26]. Table 1.1 shows scaling as a function of the number of atoms and basis set size in the calculation of Coulombic repulsion in two DFT methods [27].

[1]Scaling is represented by asymptotic notation, $O(f(x))$ [20], which summarises the computational time dependence as a function of the number of objects to analyse. However, only the function's dominant term is represented [21]. For example: $O(N^M)$ means polynomial scaling of order M [22]; $O(N!)$ stands for factorial scaling [23]; $O(C^N)$ represents exponential scaling [24]; and $O(M\log(N))$ logarithmic scaling [25].

If the algorithms used are perfect, an increase in speed necessarily means a decrease in accuracy—an increase in speed would require the use of mathematical (truncated series, averages, symmetries) and/or physical (locality constraints, potential wells, classical mechanics) shortcuts, as well as the experimental parameters—by simplifying the problem and reducing the total number of operations required [30]. For this reason, the creation of new methods, models and algorithms is of the utmost importance (see table 1.1). In fact, computational time restrictions are the main reason why semi-empirical and MD methods are widely used in chemical biology and pharmacology [31, 32].

As previously mentioned, molecular dynamics applies classical mechanics to molecular systems. In other words, they apply classical-mechanical principles to systems which require quantum mechanical ones, leading to incomplete and erroneous descriptions [33]. Cases whose scale prohibits a purely quantum treatment, but also require accurate analyses, are analysed with semi-empirical and hybrid models which can extract quantum information (often of semi-quantitative accuracy) from MD simulations. This motivated the creation of the original "quasiclassical" model [34], which has traditionally been the easiest way to extract quantum information from MD simulations. It has often (but not always) been of semi-quantitative accuracy, motivating Meyer and Miller to modify and improve it [35]. With said modifications, the Meyer-Miller model (MM), was successfully applied to three non-adiabatic electronic processes originally defined by Tully and a few problems which use the spin-boson model [1, 2].

1.2 Meyer-Miller Quasiclassical Trajectory Model with Symmetrical Windowing for Quantum States

The current iteration of Meyer and Miller's Quasiclassical Nuclear Trajectory Model (MM model) has been tested on four problems: simple avoided crossing, double avoided crossing, extended coupling and the spin-boson model for non-adiabatic dynamics of condensed-phase matter [1]. For the sake of brevity and relevance in chapter 1, it will only be explained in broad terms, a more detailed discussion can be found in chapter 2.

1.2.1 Qualitative Description of the Model

Chapter 2 provides a detailed mathematical description of the model and its modifications, for chapter 1 it is enough to describe the general algorithm.

1. Assign initial positions and states of all atoms involved.

2. Assign initial momentum and action-angle variables of all atoms involved. The initial momentum must be in the direction of the target atom(s).

3. Calculate the initial state window function.

4. Evolve the trajectories with a numerical integration routine. The integration range is defined by the user/problem.

5. Calculate the final state according to the selection criteria.

6. Calculate the final state window function.

7. Save the final state window function for later use in Monte-Carlo averaging.

8. Repeat the process a statistically sensible number of times.

9. Average the initial and final state window functions.

10. Calculate the probability of every event.

1.3 Proposal

In order to explore the workings and robustness of the MM model, it will be applied to the analysis of the of four model systems. The model will be implemented in FORTRAN 2008 and will eventually be incorporated into the non-commercial computational and quantum chemistry package CFOUR [36], primarily developed by the Stanton and Gauß research groups. It will be validated by reproducing the results obtained by Meyer and Miller [1], and used to carry out calculations for different initial states and different conditions for the spin-boson model.

The code will also be generalised as much as possible so it can eventually be added to CFOUR [36].

1.3.1 Broad Methodology

The following list of actions is the methodology with which the problem will be tackled.

1. Obtain equations of motion.

2. Write pseudo-code (list of actions). This is similar to the description given in section 1.2.1 but slightly more detailed.

3. Translate pseudo-code to FORTRAN 2008. This version of FORTRAN was chosen due to its relative ease of use, speed and ample parallelisation potential.

4. Characterise the code.
5. Validate the code by reproducing Cotton and Miller's results [1].
6. Obtain new results.
7. Generalise the code as much as possible.

1.4 Conclusions

The model has been proven to be of qualitative and quantitative accuracy for the problems defined by Tully [1, 2], and will be used to generate new results for variations of these systems.

It's worth repeating that the development of different ways of doing things is always worth a fair try. It is why one of the objectives of the present thesis is to sufficiently generalise it so it can eventually be placed at the disposition of the scientific community by including it in CFOUR.

In all, computational chemistry is a large toolbox with which we can analyse a wide range of phenomena, which would otherwise be inaccessible—or at least—extremely difficult for us to study. And thanks to the wide variety of chemical systems, there is always need to expand it. However, this is *not* the only motivator. The same drivers that haunt and propel all of mankind's endeavours: money, time, ego, efficiency and sheer curiosity; also apply to computational chemistry. For all these reasons, the search for different ways of doing things is of the utmost importance. In a sense, whether these methods work or not, is irrelevant. The fact that they exist and can be used, modified and studied, makes them invaluable for science, as they offer bifurcations in the road to a goal; bifurcations which may lead elsewhere interesting, or—at the very least—provide a different looking-glass through which to see the problem at hand.

Chapter 2

Methodology

2.1 Classical Mechanics

2.1.1 Newtonian Mechanics

Sir Isaac Newton was the first person to rigorously study the movement of classical objects by postulating three laws, the second of which is the definition of a vector quantity we call *force*:

$$F_{\mathbf{x}} = m\mathbf{a} = m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = m\frac{\mathrm{d}^2 f(\mathbf{x}, t)}{\mathrm{d}t^2} \ , \tag{2.1}$$

where m is mass, $\mathbf{x}$ represents the three spatial dimensions (x, y, z), and t is time. With this vector equation—and a few further insights—an object's trajectory can be obtained. However, this equation was later found to be incomplete and was later formalised to:

$$F_{\mathbf{x}} = \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = \frac{\mathrm{d}(m \cdot \mathbf{v})}{\mathrm{d}t} \ , \tag{2.2}$$

where $\mathbf{p}$ is momentum ($m \cdot \mathbf{v}$, where $\mathbf{v}$ is velocity in three spatial dimensions)—which is more in line with modern formulations of classical and relativistic mechanics.

Newton's formulation however, becomes very cumbersome very quickly (where dynamics are concerned), because it entails working with vectorial quantities, second spatial derivatives (accelerations), and often requires the use of geometric insights and techniques which complicate the standardisation and generalisation of problems. There are, however, two other widely used formulations, explored in sections 2.1.2 and 2.1.3.

2.1.2 Lagrangian Mechanics

The Lagrangian formalism introduces a new quantity called the *Lagrangian*:

$$\mathcal{L} = T - V \ , \tag{2.3}$$

where T is kinetic energy and V potential energy. The Lagrangian is a function of generalised positions, $\boldsymbol{q}(t)$, velocities, $\dot{\boldsymbol{q}}(t) = \frac{\mathrm{d}\boldsymbol{q}(t)}{\mathrm{d}t}$, and time, t,

$$\mathcal{L}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), t) = T(\dot{\boldsymbol{q}}(t)) - V(\boldsymbol{q}(t), t) \ . \tag{2.4}$$

The Lagrangian captures the relationship between kinetic and potential energy in a seemingly unintuitive sense. However, said relationship becomes apparent when one realises that potential energy can be converted into kinetic energy and vice-versa. Therefore, when one of them increases, the other decreases with it; this inverse relationship is captured by the Lagrangian's alternate signs.

The formalism bases itself on minimising the area/volume enclosed by the Lagrangian as it evolves in time. Said integral is called the *action*, which can be mathematically written as:

$$\mathcal{S} = \int_{t_1}^{t_2} \mathcal{L}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), t)\mathrm{d}t \ , \tag{2.5}$$

by minimising $\mathcal{S}$ we can find a particle's trajectory through space. This, however, is easier said than done, especially in the form of an integral equation. Fortunately, there exists a more tractable solution in the form of the Euler-Lagrange equation[1]. The full Euler-Lagrange equation with no restricting forces reads:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) = \frac{\partial \mathcal{L}}{\partial q_i} \ . \tag{2.6}$$

The equation applies to all generalised coordinates i, and by solving all the relevant partial differential equations one may reconstruct the particle's trajectory either as a single or multivariate function, or parametrically, whichever case is most convenient.

The Lagrangian formalism—despite being seemingly more complicated—is easier to work with than its Newtonian counterpart for complex dynamic systems, since it only requires one to have expressions for a particle's kinetic and potential energies. Furthermore, the resulting partial differential equations are often much easier to deal with, both analytically and numerically, than their Newtonian counterparts.

[1] See http://en.wikipedia.org/wiki/Euler%E2%80%93Lagrange_equation#Statement and http://en.wikipedia.org/wiki/Lagrangian_mechanics#Derivation_of_Lagrange.27s_equations for different derivations and Euler-Lagrange equation sub-types.

2.1.3 Hamiltonian Mechanics

Just as the Lagrangian formalism discards the Newtonian one's most troublesome features, the Hamiltonian formalism simplifies the Lagrangian's. By defining the generalised momentum as:

$$p_i(q_i, \dot{q}_i, t) = \frac{\partial \mathcal{L}}{\partial \dot{q}_i} \ , \tag{2.7}$$

and the Hamiltonian as the Legrende transform:

$$\mathcal{H}(\boldsymbol{q}(t), \boldsymbol{p}(t), t) = \sum_i q_i \frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \mathcal{L} = \sum_i q_i p_i - \mathcal{L} \ , \tag{2.8}$$

then, via a very simple procedure[2], one can find the following equivalences:

$$\frac{\partial \mathcal{H}}{\partial q_i} = -\dot{p}_i \quad , \quad \frac{\partial \mathcal{H}}{\partial p_i} = \dot{q}_i \quad , \quad \frac{\partial \mathcal{H}}{\partial t} = -\frac{\partial \mathcal{L}}{\partial t} \ , \tag{2.9}$$

where $\dot{p}_i = \frac{\mathrm{d}p_i}{\mathrm{d}t}$. Thus, the particle's trajectory can then be described in terms of its momenta and positions. This formalism does away with the Lagrangian's troublesome partial differential equations by converting them into a system of coupled ordinary differential equations, which are much easier to deal with—both analytically and numerically—than their Lagrangian counterparts. The main advantage of this formalism is that numerical solution of ordinary differential equations is trivial, while the same cannot be said of partial differential equations.

2.2 Quantum Mechanics

Quantum mechanics deals with the physical universe at length scales of $\lesssim 10^{-9}$ m, where gravity is eclipsed by electrostatic interactions, and where the wave-particle duality becomes apparent. Quantum mechanics is governed by the time-independent eq. (2.10), and time-dependent eq. (2.11) Schrödinger Equations:

$$\hat{H}\Psi(\boldsymbol{r}) = E\Psi \tag{2.10a}$$

$$\hat{H} = \frac{-\hbar^2}{2\mu}\nabla^2 + V(\boldsymbol{r}) \tag{2.10b}$$

$$i\hbar\frac{\partial}{\partial t}\Psi(\boldsymbol{r}, t) = \hat{H}\Psi(\boldsymbol{r}, t) \tag{2.11a}$$

$$\hat{H} = \frac{-\hbar^2}{2\mu}\nabla^2 + V(\boldsymbol{r}, t) \ , \tag{2.11b}$$

[2] See http://en.wikipedia.org/wiki/Hamiltonian_mechanics#Deriving_Hamilton.27s_equations for the derivation of the equivalences in eq. (2.9).

where ∇^2 is the Laplacian operator, $\hbar$ the reduced planck constant, μ the particle's reduced mass, V the potential, and Ψ the wavefunction.

However, there are many challenges which come with such simple-looking equations. For starters, one must solve for the wavefunction, which is only analytically solvable for very simple systems—the most complex of being the hydrogen atom. For more complex systems, it must be constructed iteratively from basis sets. The other big problem is the quickly increasing complexity of the Hamiltonian operator, which must include terms for the nuclear and electronic kinetic energies ($\hat{T}$), electron-nucleus (en), electron-electron (ee) and nucleus-nucleus (nn) electrostatic interactions ($\hat{V}$) found in eq. (2.12), as well as various smaller terms:

$$\hat{T}_n = -\sum_i \frac{\hbar^2}{2M_i}\nabla^2_{\boldsymbol{R}_i} \tag{2.12a}$$

$$\hat{T}_e = -\sum_i \frac{\hbar^2}{2m_e}\nabla^2_{\boldsymbol{r}_i} \tag{2.12b}$$

$$\hat{V}_{en} = -\sum_i \sum_j \frac{Z_i e^2}{4\pi\epsilon_0 |\boldsymbol{R}_i - \boldsymbol{r}_j|} \tag{2.12c}$$

$$\hat{V}_{ee} = \sum_i \sum_{j>i} \frac{e^2}{4\pi\epsilon_0 |\boldsymbol{r}_i - \boldsymbol{r}_j|} \tag{2.12d}$$

$$\hat{V}_{nn} = \sum_i \sum_{j>i} \frac{Z_i Z_j e^2}{4\pi\epsilon_0 |\boldsymbol{R}_i - \boldsymbol{R}_j|}\ , \tag{2.12e}$$

where $\boldsymbol{R}$ is the nuclear coordinate, $\boldsymbol{r}$ the electronic coordinate, M the atomic mass, m the electron mass, Z the atomic number, e the electric charge, ϵ_0 the vacuum permittivity, and i, j labels for electrons and nuclei.

The difficulty with working with such a Hamiltonian operator is clearly evident. The problem worsens when it is required to be time-dependent, such as in the case of atomic trajectories, where the Born-Oppenheimer approximation[3] no longer applies. For such cases, the wavefunction must either be constructed or evolved at each time step, rendering the procedure prohibitively slow.

There are however, some problems which require us to do exactly this. For simple unphysical systems, it's not too difficult to do so quantum mechanically, but as the system complexity increases, things become much too complicated for this to be viable. Systems such as vibronic structures, catalytic cycles, enzymatic activity, and other systems where trajectories and electronic states are strongly coupled to one another. This is where hybrid models come into the picture. One of which, is the Meyer-Miller Quasiclassical Trajectory (MM) Model explored herein.

[3]The nuclei are taken to be stationary with respect to the electrons, eliminating the nuclear kinetic energy term and simplifying many interaction calculations.

2.3 Meyer-Miller Quasiclassical Trajectory Model with Symmetrical Windowing for Quantum States

Quasiclassical and semiclassical models scale better than their purely quantum counterparts, and are more accurate than pure molecular dynamics. Unlike quantum methods, they don't build molecular wavefunctions, but often include mean-field approximations of electronic information in the form of electronic 'action-angle' variables, and potential energy surfaces (PES); thus providing an effective potential acting on the particle, and a way for trajectories and electronic states to couple with each other. The MM model works in such a way.

2.3.1 Description and General Characteristics

The Meyer-Miller Quasiclassical Hamiltonian is a classical analogue of a time-dependent version of the molecular Hamiltonian operator. However, as stated in section 2.2, this is extremely impractical. This can be extremely time consuming, and scaling issues only add to the problem. Which is why creating and exploring quasiclassical approaches is a worthwhile endeavour.

Many quasiclassical approaches are defined in terms of so-called 'action-angle' variables and the Meyer-Miller model is no exception. In action-angle variables, the diabatic Meyer-Miller Hamiltonian [1] reads:

$$H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{n},\boldsymbol{q}) = \frac{\boldsymbol{P}^2}{2\mu} + \sum_{k=1}^{F} n_k H_{kk}(\boldsymbol{R}) + 2\sum_{k<k'=1}^{F} \sqrt{(n_k+\gamma)(n_{k'}+\gamma)} \times \cos(q_k - q_{k'}) H_{kk'}(\boldsymbol{R}) \ . \tag{2.13}$$

However, it is more convenient to change from action-angle variables to Cartesian variables $\{p_k, x_k\}$ via the canonical transformation:

$$x_k = \sqrt{2(n_k+\gamma)}\cos(q_k) \tag{2.14a}$$

$$p_k = -\sqrt{2(n_k+\gamma)}\sin(q_k) \ , \tag{2.14b}$$

because this approach eliminates the need to evaluate trigonometric functions during numerical integration, and makes the derivation of the equations of motion a less dolorous affair.

The initial conditions are set within the action-angle frame using Monte-Carlo (MC) sampling methods and the following formulae:

$$n_k(0) = N_k + \gamma(2 \cdot RN_k - 1) \tag{2.15a}$$

$$q_k(0) = 2\pi \cdot RN'_k \ , \tag{2.15b}$$

where $N_k = 0$ means the k^{th} state is unoccupied, and $N_k = 1$ means the k^{th} state is occupied, γ is the zero-point energy parameter, and finally RN_k and RN'_k are two uniformly distributed random numbers $\in [0, 1]$ interval.

After all trajectories have been integrated (using the traditional Runge-Kutta-4-Gill method), eq. (2.16) (obtained from eq. (2.14)) can be used to calculate the final values of $n_k(t)$:

$$n_k = \frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \gamma \ , \tag{2.16}$$

(the angle variable is irrelevant).

A distinguishing feature of the version of the MM model explored here is the symmetrical windowing of the electronic states in terms of the Heaviside function, $h(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$:

$$W_{N_k}(n_k) = \frac{1}{\Delta n} h\left(\frac{\Delta n}{2} - |n_k - N_k|\right) \ . \tag{2.17}$$

In order for $W_{N_k} \neq 0$, $n_k \in [N_k - \Delta n/2,\ N_k + \Delta n/2]$. Thus making Δn the function's width—which in this case is defined as, $\Delta n = 2\gamma$. It is worth noting that the window function for a state k state is the product of such functions for each electronic degree of freedom. In the case of the sample problems there are two electronic states ($F = 2$). For the first one, $(N_1, N_2) = (1, 0)$, the window function is, $W_1(n_1) \cdot W_0(n_2)$; similarly for the second state, $(N_1, N_2) = (0, 1)$ we have, $W_0(n_1) \cdot W_1(n_2)$.

A trajectory is a assigned a final state k if the final values of n_k comply with the following inequality:

$$N_k - \gamma \leq n_k \leq N_k + \gamma \ , \tag{2.18}$$

in terms of Cartesian coordinates this means:

$$N_k \leq \frac{1}{2}x_k^2 + \frac{1}{2}p_k^2 \leq N_k + 2\gamma \ . \tag{2.19}$$

It's worth noting that the criteria must be simultaneously met for all k electronic degrees of freedom—for when they are not, we get a 'mixture' of states, which is incompatible with the model, and the calculation has to be restarted until the initial conditions allow them to be met.

In certain problems, care must also be taken to check whether the particle has been transmitted i.e. has left the integration area *opposite* to its initial side, or reflected i.e. has left integration area from the *same* side to where it started. It is of the utmost importance to take this into account when tackling problems where particle trajectories, as well as transition probabilities interest us.

The final window functions are then calculated via eqs. (2.17) and (2.19), and Monte-Carlo (MC) averaged. Lastly, the transition probabilities from initial to final state, $P_{f\leftarrow i}$, are calculated by multiplying the initial and final window functions (the *and* rule of probability), and dividing by the sum of the corresponding quantities for all possible final states,

$$P_{f\leftarrow i} = \frac{\left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle}{\sum_{k=1}^{F} \left\langle \prod_{f=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle} \,. \tag{2.20}$$

As previously stated, in cases where reflection and transmission are also important, eq. (2.20) changes to:

$$P^{a}_{f\leftarrow i} = \frac{\left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle_a}{\sum_{a=r,\ t} \left(\sum_{f=1}^{F} \left\langle \prod_{k=1}^{F} W_{\delta_{fk}}(n_k(t)) \cdot \prod_{k=1}^{F} W_{\delta_{ik}}(n_k(0)) \right\rangle_a \right)} \,, \tag{2.21}$$

where $\delta_{ij} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$, $\langle \cdots \rangle$ denotes Monte-Carlo averaging, $r \equiv$ reflection and $t \equiv$ transmission. The implementation of both procedures does not differ much, the major difference is that in the case of eq. (2.20), the numerator can be implemented as a single 1D array with F items; in the case of eq. (2.21) it can be implemented as two (one for reflection and one for transmission) 1D arrays with F items, a 2D array with $2 \times F$ items, or a 1D array with $2F$ items. Of course, whichever method of implementation is chosen, care must be taken to make sure the appropriate element is being read, which is why the present implementation bases itself on the safe option: two 1D arrays with F items.

2.3.2 Diabatic Hamiltonian

There are two MM Hamiltonians, for diabatic systems—whose conditions vary so quickly that the system has no time to adjust to them—the Hamiltonian in cartesian

coordinates[4] reads:

$$H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{p},\boldsymbol{x}) = \frac{\boldsymbol{P}^2}{2\mu} + \bar{H}(\boldsymbol{R}) + \sum_{k<k'=1}^{F} \left\{ \begin{array}{r} \frac{1}{F}(H_{kk}(\boldsymbol{R}) - H_{k'k'}(\boldsymbol{R})) \cdot \left(\frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \frac{1}{2}p_{k'}^2 - \frac{1}{2}x_{k'}^2\right) \\ +H_{kk'}(\boldsymbol{R}) \cdot (p_k p_{k'} + x_k x_{k'}) \end{array} \right\} \tag{2.22a}$$

$$\bar{H}(\boldsymbol{R}) = \frac{1}{F}\sum_{k=1}^{F} H_{kk}(\boldsymbol{R}) \ . \tag{2.22b}$$

For $F = 2$ eq. (2.22) becomes eq. (2.23):

$$\begin{aligned} H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{p},\boldsymbol{x}) = \frac{\boldsymbol{P}^2}{2\mu} &+ \frac{1}{2}(H_{11}(\boldsymbol{R}) + H_{22}\boldsymbol{R}) \\ &+ \frac{1}{2}(H_{11}(\boldsymbol{R}) - H_{22}(\boldsymbol{R})) \cdot \left(\frac{1}{2}p_1^2 + \frac{1}{2}x_1^2 - \frac{1}{2}p_2^2 - \frac{1}{2}x_2^2\right) \\ &+ H_{12}(\boldsymbol{R}) \cdot (p_1 p_2 + x_1 x_2) \ . \end{aligned} \tag{2.23}$$

Equations of Motion

The equations of motion are obtained in the manner described by eq. (2.9). For notation purposes, the Hamiltonian's and matrix elements' dependencies will not be shown (see eq. (2.22)). With $i \in [1, F]$, the general equations of motion are:

$$\dot{\boldsymbol{R}} = \frac{\partial H}{\partial \boldsymbol{P}} = \frac{\boldsymbol{P}}{\mu} \tag{2.24a}$$

$$\dot{\boldsymbol{P}} = -\frac{\partial H}{\partial \boldsymbol{R}} = -\frac{\partial \bar{H}}{\partial \boldsymbol{R}} - \sum_{k<k'=1}^{F} \left\{ \begin{array}{r} \frac{1}{2F}\left(\frac{\partial H_{kk}}{\partial \boldsymbol{R}} - \frac{\partial H_{k'k'}}{\partial \boldsymbol{R}}\right) \cdot (p_k^2 + x_k^2 - p_{k'}^2 - x_{k'}^2) \\ +\frac{\partial H_{kk'}}{\partial \boldsymbol{R}}(p_k p_{k'} + x_k x_{k'}) \end{array} \right\} \tag{2.24b}$$

$$\dot{x}_i = \frac{\partial H}{\partial p_i} = \sum_{k<k'=1}^{F} \left\{ \frac{1}{2F}(H_{kk} - H_{k'k'})\frac{\partial (p_k^2 - p_{k'}^2)}{\partial p_i} + H_{kk'}\frac{\partial (p_k p_{k'})}{\partial p_i} \right\} \tag{2.24c}$$

$$\dot{p}_i = -\frac{\partial H}{\partial x_i} = -\sum_{k<k'=1}^{F} \left\{ \frac{1}{2F}(H_{kk} - H_{k'k'})\frac{\partial (x_k^2 - x_{k'}^2)}{\partial x_i} + H_{kk'}\frac{\partial (x_k x_{k'})}{\partial x_i} \right\} \ . \tag{2.24d}$$

[4] For details about the transformation see appendix B.1.

For $F = 2$ they are:

$$\dot{\boldsymbol{R}} = \frac{\boldsymbol{P}}{\mu} \tag{2.25a}$$

$$\dot{\boldsymbol{P}} = -\frac{1}{2}\left(\frac{\partial H_{11}}{\partial \boldsymbol{R}} + \frac{\partial H_{22}}{\partial \boldsymbol{R}}\right) - \frac{1}{4}\left(\frac{\partial H_{11}}{\partial \boldsymbol{R}} - \frac{\partial H_{22}}{\partial \boldsymbol{R}}\right) \cdot (p_1^2 + x_1^2 - p_2^2 - x_2^2) - \frac{\partial H_{12}}{\partial \boldsymbol{R}}(p_1 p_2 + x_1 x_2) \tag{2.25b}$$

$$\dot{x}_1 = \frac{1}{2} p_1 (H_{11} - H_{22}) + p_2 H_{12} \tag{2.25c}$$

$$\dot{p}_1 = -\frac{1}{2} x_1 (H_{11} - H_{22}) - x_2 H_{12} \tag{2.25d}$$

$$\dot{x}_2 = -\frac{1}{2} p_2 (H_{11} - H_{22}) + p_1 H_{12} \tag{2.25e}$$

$$\dot{p}_2 = \frac{1}{2} x_2 (H_{11} - H_{22}) - x_1 H_{12} \ . \tag{2.25f}$$

2.3.3 Adiabatic Hamiltonian

For adiabatic systems—whose conditions vary slowly enough that the system can adjust to them—the Hamiltonian is treated in exactly the same way, but being a diagonalised version of the adiabatic Hamiltonian, lacks an explicit non-diagonal term, uses the eigenvalues of the adiabatic PES, E_k, and contains a so-called 'mixing angle' term, $\omega(\boldsymbol{R})$, in the nuclear kinetic energy term:

$$H(\boldsymbol{P}, \boldsymbol{R}, \boldsymbol{p}, \boldsymbol{x}) = \frac{|\boldsymbol{P} + \nabla \boldsymbol{P}|^2}{2\mu} + \bar{E}(\boldsymbol{R}) + \sum_{k<k'=1}^{F} \frac{1}{F}(E_k(\boldsymbol{R}) - E_{k'}(\boldsymbol{R})) \cdot \left(\frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \frac{1}{2}p_{k'}^2 - \frac{1}{2}x_{k'}^2\right) \tag{2.26a}$$

$$\bar{E}(\boldsymbol{R}) = \frac{1}{F}\sum_{k=1}^{F} E_k(\boldsymbol{R}) \ , \tag{2.26b}$$

where,

$$\nabla \boldsymbol{P} = \sum_{k<k'=1}^{F} \omega(\boldsymbol{R}) \cdot (p_k x_{k'} - p_{k'} x_k) \tag{2.27a}$$

$$\Delta \boldsymbol{P} = \sum_{k<k'=1}^{F} \frac{\partial \omega(\boldsymbol{R})}{\partial \boldsymbol{R}} \cdot (p_k x_{k'} - p_{k'} x_k) \tag{2.27b}$$

$$\omega(\boldsymbol{R}) = \frac{1}{2} \arctan\left(\frac{2H_{kk'}(\boldsymbol{R})}{H_{kk}(\boldsymbol{R}) - H_{k'k'}(\boldsymbol{R})}\right) \ . \tag{2.27c}$$

For $F = 2$ eq. (2.26) becomes eq. (2.28):

$$H(\boldsymbol{P}, \boldsymbol{R}, \boldsymbol{p}, \boldsymbol{x}) = \frac{|\boldsymbol{P} + \nabla \boldsymbol{P}|^2}{2\mu} + \frac{1}{2}(E_1 + E_2) + \frac{1}{2}(E_1(\boldsymbol{R}) - E_2(\boldsymbol{R})) \cdot \left(\frac{1}{2}p_1^2 + \frac{1}{2}x_1^2 - \frac{1}{2}p_2^2 - \frac{1}{2}x_2^2\right) . \tag{2.28}$$

Equations of Motion

For notation purposes, the Hamiltonian's and matrix elements' dependencies will not be shown (see eq. (2.26)). With $i \in [1, F]$, the general equations of motion are:

$$\dot{\boldsymbol{R}} = \frac{\partial H}{\partial \boldsymbol{P}} = \frac{\boldsymbol{P} + \nabla \boldsymbol{P}}{\mu} \tag{2.29a}$$

$$\dot{\boldsymbol{P}} = -\frac{\partial H}{\partial \boldsymbol{R}} = -\frac{\boldsymbol{P} + \nabla \boldsymbol{P}}{\mu} \cdot \Delta \boldsymbol{P} - \frac{\partial \bar{E}}{\partial \boldsymbol{R}} - \sum_{k<k'=1}^{F} \left\{ \frac{1}{2F} \left(\frac{\partial E_k}{\partial \boldsymbol{R}} - \frac{\partial E_{k'}}{\partial \boldsymbol{R}} \right) \cdot (p_k^2 + x_k^2 - p_{k'}^2 - x_{k'}^2) \right\} \tag{2.29b}$$

$$\dot{x}_i = \frac{\partial H}{\partial p_i} = \frac{\boldsymbol{P} + \nabla \boldsymbol{P}}{\mu} \cdot \frac{\partial \nabla \boldsymbol{P}}{\partial p_i} + \sum_{k<k'=1}^{F} \left\{ \frac{1}{2F}(E_k - E_{k'}) \frac{\partial (p_k^2 - p_{k'}^2)}{\partial p_i} \right\} \tag{2.29c}$$

$$\dot{p}_i = -\frac{\partial H}{\partial x_i} = -\frac{\boldsymbol{P} + \nabla \boldsymbol{P}}{\mu} \cdot \frac{\partial \nabla \boldsymbol{P}}{\partial x_i} - \sum_{k<k'=1}^{F} \left\{ \frac{1}{2F}(E_k - E_{k'}) \frac{\partial (x_k^2 - x_{k'}^2)}{\partial x_i} \right\} . \tag{2.29d}$$

For $F = 2$ and after substituting for $\nabla \boldsymbol{P}$ and $\Delta \boldsymbol{P}$ they read:

$$\dot{\boldsymbol{R}} = \frac{\boldsymbol{P}}{\mu} + \frac{1}{2\mu} \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right)(p_1 x_2 - p_2 x_1) \tag{2.30a}$$

$$\begin{aligned}\dot{\boldsymbol{P}} = &- \left[\frac{\boldsymbol{P}}{\mu} + \frac{1}{2\mu} \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right)(p_1 x_2 - p_2 x_1)\right] \\ &\times \frac{\left[\frac{\partial H_{12}}{\partial \boldsymbol{R}}(H_{11} - H_{22}) + H_{12}\left(\frac{\partial H_{22}}{\partial \boldsymbol{R}} - \frac{\partial H_{11}}{\partial \boldsymbol{R}}\right)\right](p_1 x_2 - p_2 x_1)}{4H_{12}^2 + (H_{11} - H_{22})^2} \\ &- \frac{1}{4}\left(\frac{\partial E_1}{\partial \boldsymbol{R}} - \frac{\partial E_2}{\partial \boldsymbol{R}}\right)(p_1^2 + x_1^2 - p_2^2 - x_2^2) - \frac{1}{2}\left(\frac{\partial E_1}{\partial \boldsymbol{R}} + \frac{\partial E_2}{\partial \boldsymbol{R}}\right)\end{aligned} \tag{2.30b}$$

$$\begin{aligned}\dot{x}_1 = &\left[\frac{\boldsymbol{P}}{2\mu} + \frac{1}{4\mu} \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right)(p_1 x_2 - p_2 x_1)\right] x_2 \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right) \\ &+ \frac{p_1}{2}(E_1 - E_2)\end{aligned} \tag{2.30c}$$

$$\begin{aligned}\dot{p}_1 = &\left[\frac{\boldsymbol{P}}{2\mu} + \frac{1}{4\mu} \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right)(p_1 x_2 - p_2 x_1)\right] p_2 \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right) \\ &- \frac{x_1}{2}(E_1 - E_2)\end{aligned} \tag{2.30d}$$

$$\begin{aligned}\dot{x}_2 = &- \left[\frac{\boldsymbol{P}}{2\mu} + \frac{1}{4\mu} \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right)(p_1 x_2 - p_2 x_1)\right] x_1 \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right) \\ &- \frac{p_2}{2}(E_1 - E_2)\end{aligned} \tag{2.30e}$$

$$\begin{aligned}\dot{p}_2 = &- \left[\frac{\boldsymbol{P}}{2\mu} + \frac{1}{4\mu} \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right)(p_1 x_2 - p_2 x_1)\right] p_2 \arctan\left(\frac{2H_{12}}{H_{11} - H_{22}}\right) \\ &+ \frac{x_2}{2}(E_1 - E_2)\ .\end{aligned} \tag{2.30f}$$

2.4 Model Problems

All problems analysed in this work have only two electronic degrees of freedom (electronic states), $\therefore F = 2$. Furthermore, all problems assume $\mu = m_k = 2000$ and all units are in atomic units.

Tully analysed three avoided crossing problems with his least-hops algorithm [2]. These three problems, along with two others which use the spin-boson model for condensed-phase dynamics were analysed here—all of which were analysed for initial states 1 and 2. It is worth noting that the smaller the difference between two adiabatic PES becomes, the likelier an electronic transition from one to the other becomes.

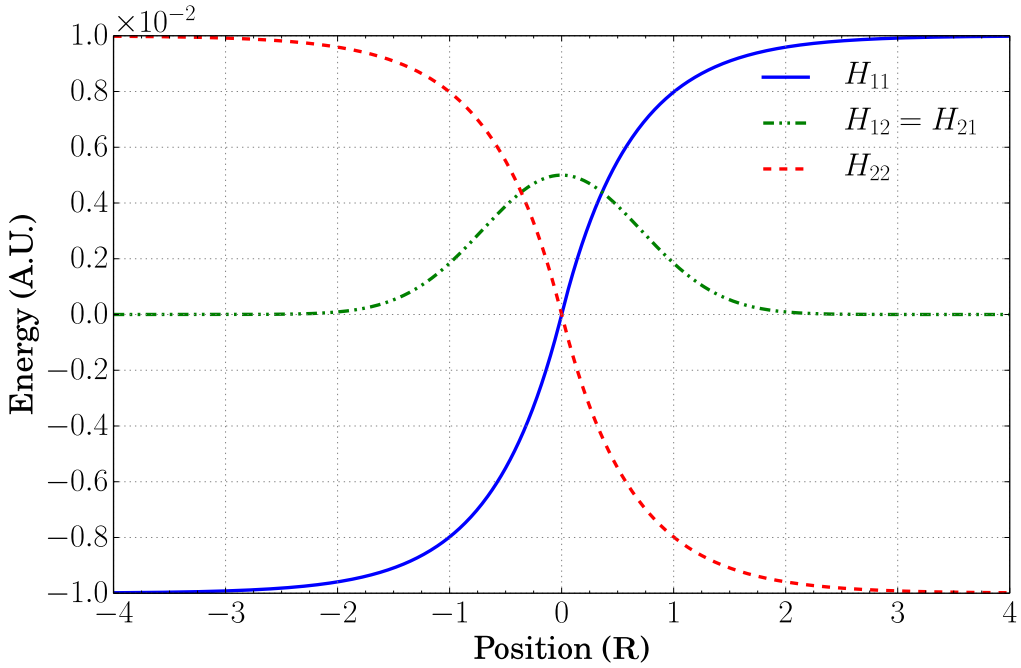


Figure 2.1: Diabatic PES.

2.4.1 Single Avoided Crossing

Potential Energy Surfaces

The diabatic PES for the single avoided crossing were defined by Tully [2] as:

$$H_{11}(R) = A \begin{cases} \left(1 - e^{-BR}\right) & R \geq 0 \\ -\left(1 - e^{BR}\right) & R < 0 \end{cases} \tag{2.31a}$$

$$H_{22}(R) = -H_{11}(R) \tag{2.31b}$$

$$H_{12}(R) = H_{21}(R) = Ce^{-DR^2} \ , \tag{2.31c}$$

where $A = 0.01,\ B = 1.6,\ C = 0.005,$ and $D = 1$. They are shown in fig. 2.1.

The equations of motion (eq. (2.24)) require the calculation of gradients for all relevant PES. For our purposes, they can be obtained analytically eq. (2.32), reducing computational demands. Less ideal systems would require numerical differentiation. The diabatic PES derivatives are:

$$\frac{\partial H_{11}}{\partial R} = AB \begin{cases} e^{-BR} & R \geq 0 \\ e^{BR} & R < 0 \end{cases} \tag{2.32a}$$

$$\frac{\partial H_{22}}{\partial R} = -\frac{\partial H_{11}}{\partial R} \tag{2.32b}$$

$$\frac{\partial H_{12}}{\partial R} = \frac{\partial H_{21}}{\partial R} = -2CDe^{-DR^2}R \ . \tag{2.32c}$$

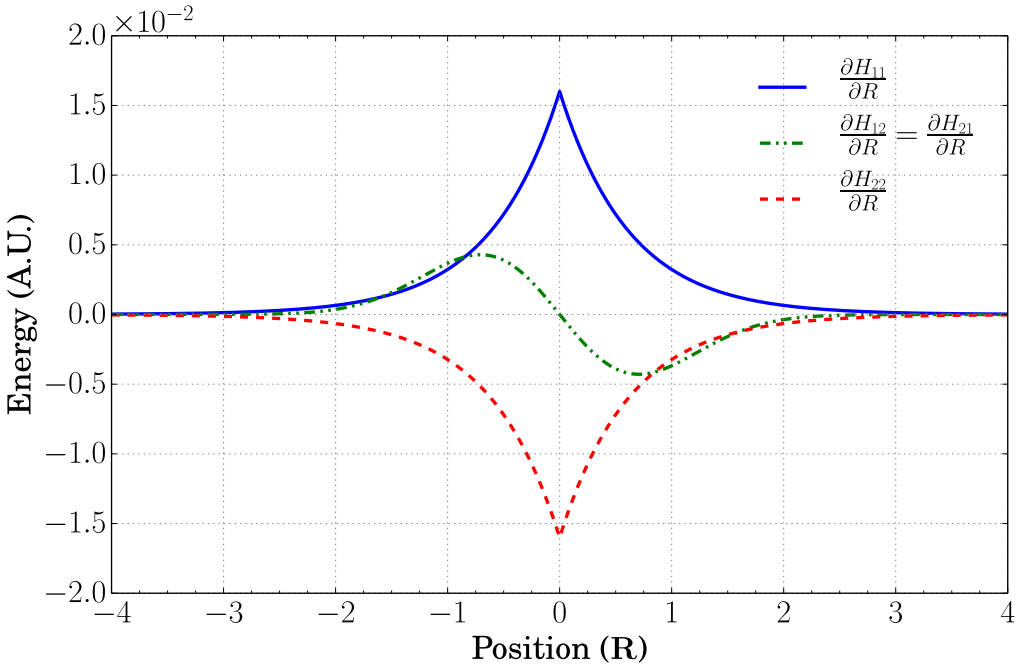


Figure 2.2: Diabatic PES derivatives.

They are shown in fig. 2.2.

The adiabatic PES (E_i) are the eigenvalues of the diabatic PES Hamiltonian matrix, which are obtained through eigenvalue decomposition. In our case, they can be obtained analytically, but less ideal systems require numerical methods. For the single avoided crossing, these are,

$$E_1 = -e^{-DR^2}\sqrt{C^2 + A^2 e^{2DR^2}\begin{cases}\left(1-e^{-BR}\right)^2 & R \geq 0\\ \left(1-e^{BR}\right)^2 & R < 0\end{cases}} \tag{2.33a}$$

$$E_2 = -E_1\ . \tag{2.33b}$$

The diabatic Hamiltonian eq. (2.22) does not require the calculation of adiabatic PES, or their derivatives. However, it is often useful to visualise the adiabatic PES because they can provide us with useful physical insight. They are shown in fig. 2.3. Their derivatives are not shown here because they are irrelevant.

The coupling between electronic states—and therefore the probability of transition—is at its greatest where there the energy difference between both adiabatic PES is at a minimum, as shown in fig. 2.4.

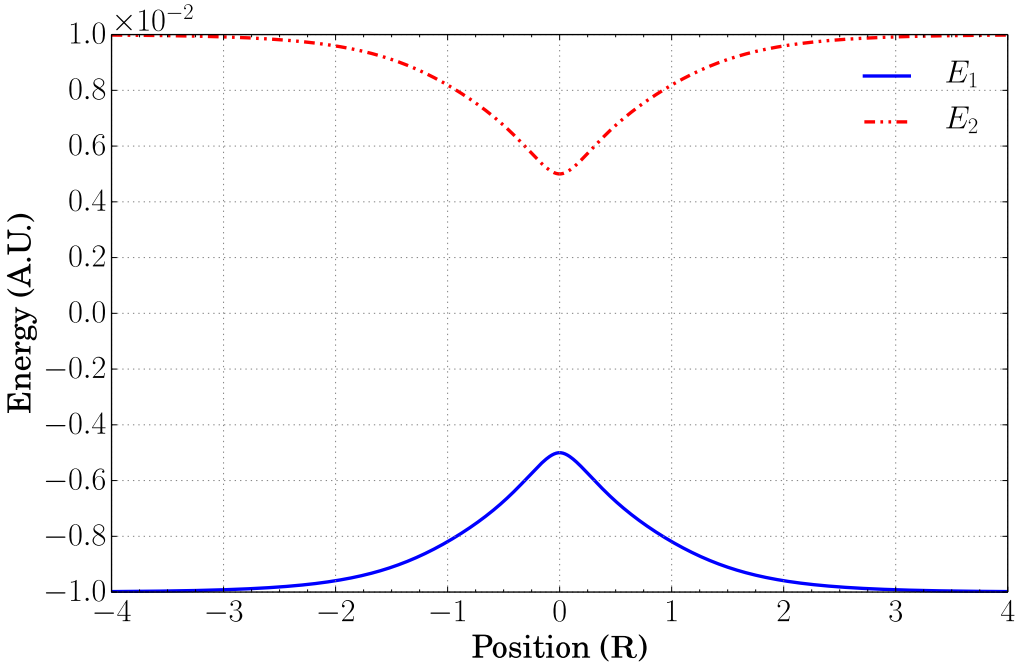


Figure 2.3: Adiabatic PES. Eigenvalues of the diabatic Hamiltonian matrix.

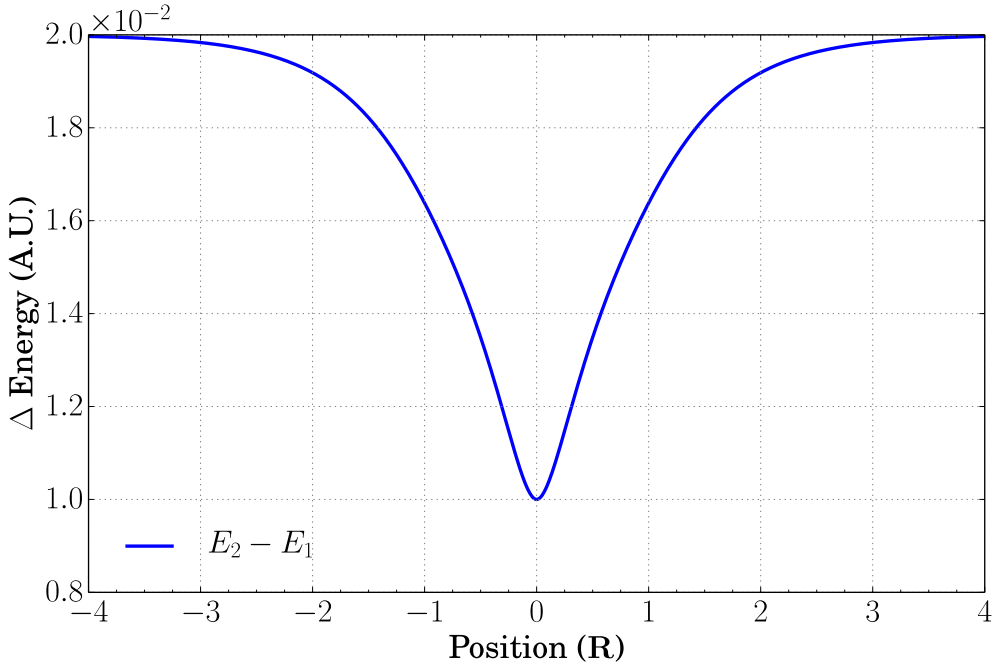


Figure 2.4: Energy difference between adiabatic PES.

Equations of Motion

For the single avoided crossing, the equations of motion are:

$$\dot{R} = \frac{P}{\mu} \tag{2.34a}$$

$$\dot{P} = 2CDRe^{-DR^2}(p_1p_2 + x_1x_2) - \frac{1}{2}\left(p_1^2 + x_1^2 - p_2^2 - x_2^2\right) AB \begin{cases} e^{-BR} & R \geq 0 \\ e^{BR} & R < 0 \end{cases} \tag{2.34b}$$

$$\dot{x}_1 = Cp_2e^{-DR^2} + p_1A \begin{cases} (1 - e^{-BR}) & R \geq 0 \\ (e^{BR} - 1) & R < 0 \end{cases} \tag{2.34c}$$

$$\dot{p}_1 = -Cx_2e^{-DR^2} - x_1A \begin{cases} (1 - e^{-BR}) & R \geq 0 \\ (e^{BR} - 1) & R < 0 \end{cases} \tag{2.34d}$$

$$\dot{x}_2 = Cp_1e^{-DR^2} - p_2A \begin{cases} (1 - e^{-BR}) & R \geq 0 \\ (e^{BR} - 1) & R < 0 \end{cases} \tag{2.34e}$$

$$\dot{p}_2 = -Cx_1e^{-DR^2} + x_2A \begin{cases} (1 - e^{-BR}) & R \geq 0 \\ (e^{BR} - 1) & R < 0 \end{cases} . \tag{2.34f}$$

2.4.2 Double Avoided Crossing

Potential Energy Surfaces

The diabatic PES for the double avoided crossing were defined by Tully [2] as:

$$H_{11}(R) = 0 \tag{2.35a}$$

$$H_{22}(R) = -Ae^{-BR^2} + E_0 \tag{2.35b}$$

$$H_{12}(R) = H_{21}(R) = Ce^{-DR^2} , \tag{2.35c}$$

where $A = 0.1,\ B = 0.28,\ C = 0.015,\ D = 0.06,$ and $E_0 = 0.05$. They are shown in fig. 2.5.

Their derivatives are:

$$\frac{\partial H_{11}}{\partial R} = 0 \tag{2.36a}$$

$$\frac{\partial H_{22}}{\partial R} = 2ABe^{-BR^2}R \tag{2.36b}$$

$$\frac{\partial H_{12}}{\partial R} = \frac{\partial H_{21}}{\partial R} = -2CDe^{-DR^2}R . \tag{2.36c}$$

They are shown in fig. 2.6.

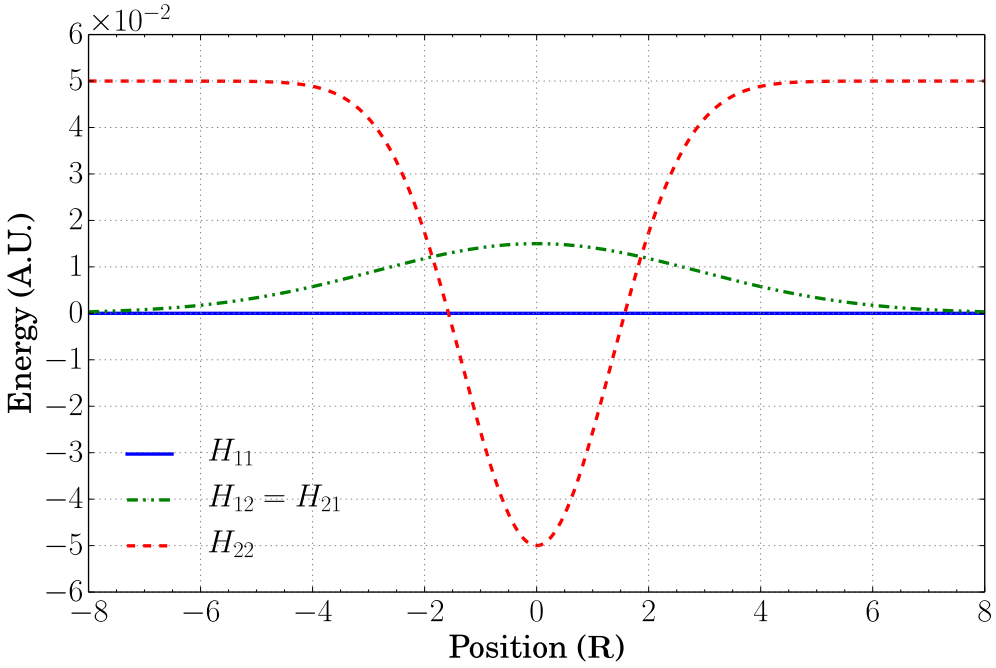


Figure 2.5: Diabatic PES.

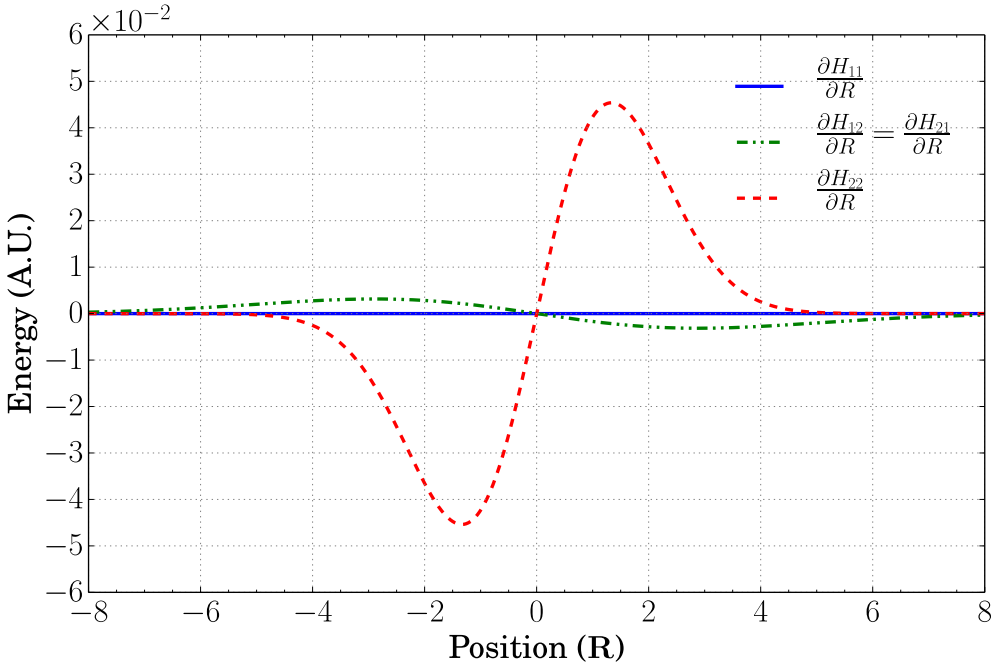


Figure 2.6: Diabatic PES derivatives.

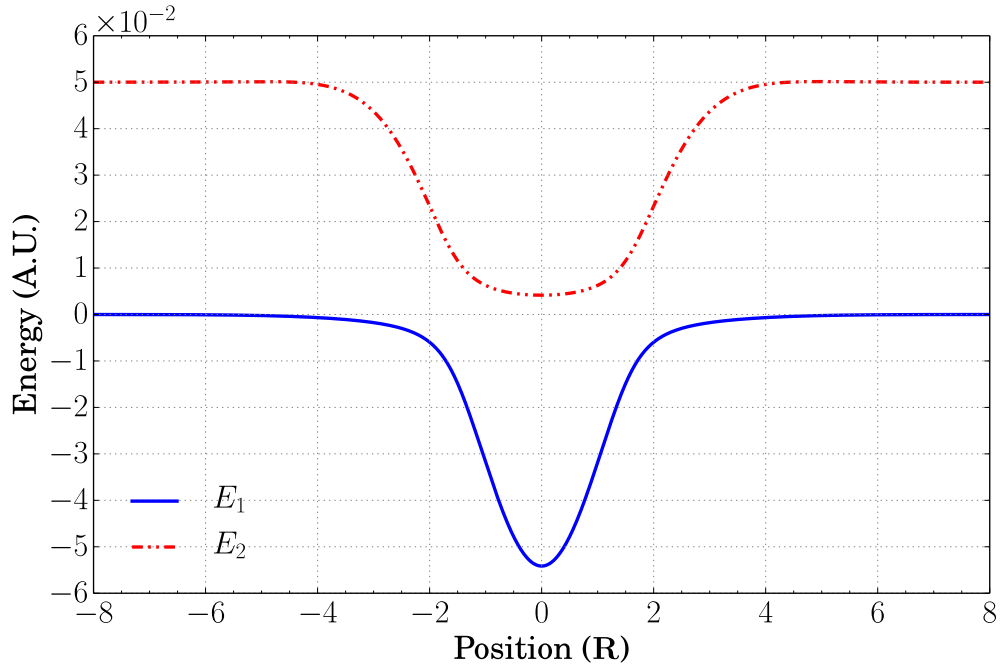


Figure 2.7: Adiabatic PES. Eigenvalues of the diabatic Hamiltonian matrix.

The adiabatic PES are:

$$E_1 = \frac{1}{2}e^{-(B+D)R^2}\left(-Ae^{DR^2} + e^{(B+D)R^2}E_0 - \sqrt{4C^2e^{2BR^2} + e^{2DR^2}\left(A - e^{BR^2}E_0\right)^2}\right) \tag{2.37a}$$

$$E_2 = \frac{1}{2}e^{-(B+D)R^2}\left(-Ae^{DR^2} + e^{(B+D)R^2}E_0 + \sqrt{4C^2e^{2BR^2} + e^{2DR^2}\left(A - e^{BR^2}E_0\right)^2}\right) . \tag{2.37b}$$

They are shown in fig. 2.7

In this case, there are two minimum energy gaps between both adiabatic PES, shown in fig. 2.8. This brings about some rather interesting behaviours shown in section 3.2.

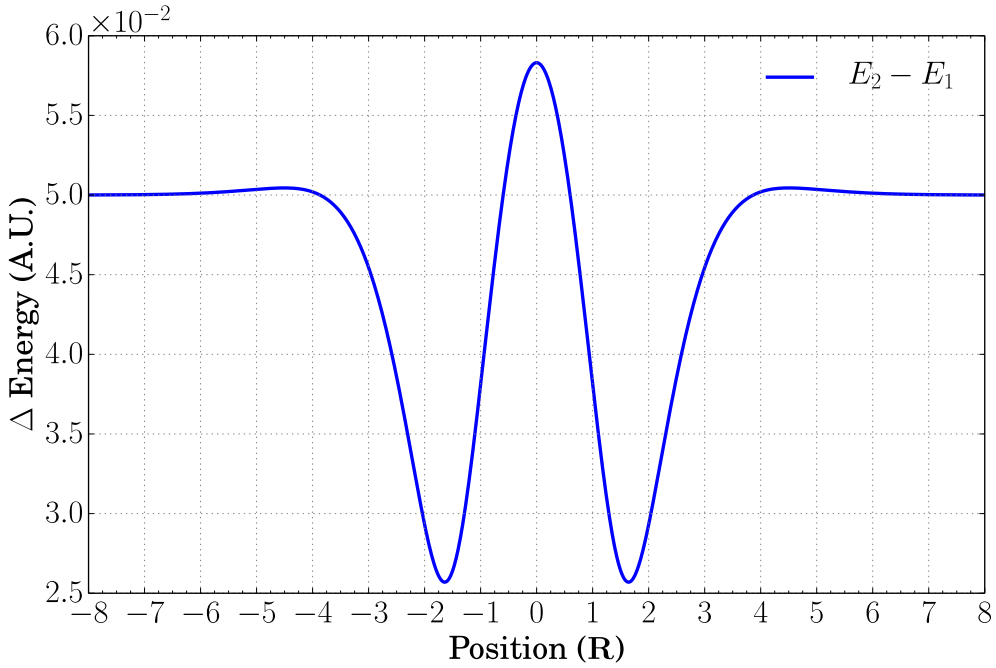


Figure 2.8: Energy difference between adiabatic PES.

Equations of Motion

For the double avoided crossing, the equations of motion are:

$$\dot{R} = \frac{P}{\mu} \tag{2.38a}$$

$$\dot{P} = R\left(ABe^{-BR^2}\left(\frac{p_1^2}{2} - \frac{p_2^2}{2} + \frac{x_1^2}{2} - \frac{x_2^2}{2} - 1\right) + 2CDe^{-DR^2}(p_1p_2 + x_1x_2)\right) \tag{2.38b}$$

$$\dot{x}_1 = \frac{1}{2}p_1\left(Ae^{-BR^2} - E_0\right) + Cp_2e^{-DR^2} \tag{2.38c}$$

$$\dot{p}_1 = -\frac{1}{2}x_1\left(Ae^{-BR^2} - E_0\right) - Cx_2e^{-DR^2} \tag{2.38d}$$

$$\dot{x}_2 = -\frac{1}{2}p_2\left(Ae^{-BR^2} - E_0\right) + Cp_1e^{-DR^2} \tag{2.38e}$$

$$\dot{p}_2 = \frac{1}{2}x_2\left(Ae^{-BR^2} - E_0\right) - Cx_1e^{-DR^2}\ . \tag{2.38f}$$

2.4.3 Extended Coupling

Potential Energy Surfaces

This problem has non-vanishing non-diagonal elements in the Hamiltonian matrix. Said elements are perturbations from a pure state, and when they do not vanish as

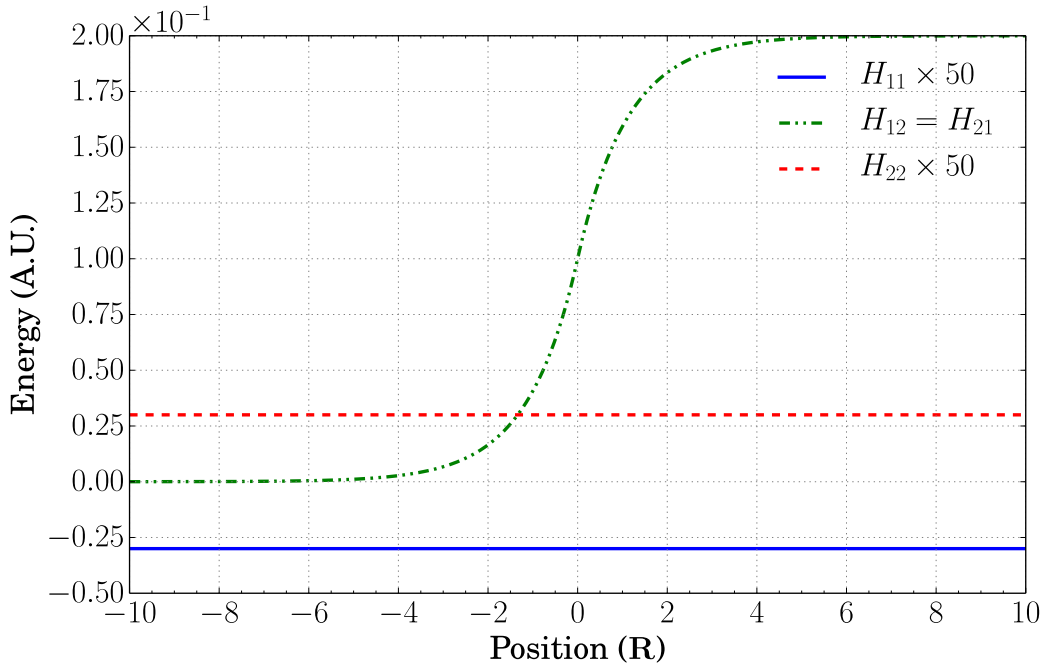


Figure 2.9: Diabatic PES.

$R \to \pm\infty$, one must use the adiabatic Hamiltonian. The diagonal elements of the Hamiltonian matrix are also constant, lending further arguments to the use of the adiabatic Hamiltonian due to the system's relatively unchanging conditions.

The diabatic PES for the extended coupling problem were defined by Tully [2] as:

$$H_{11}(R) = -A \tag{2.39a}$$

$$H_{22}(R) = -H_{11} \tag{2.39b}$$

$$H_{12}(R) = H_{21}(R) = B \begin{cases} \left(2 - e^{-CR}\right) & R \geq 0 \\ e^{CR} & R < 0 \end{cases}, \tag{2.39c}$$

where $A = 6 \times 10^{-4},\ B = 0.1,$ and $C = 0.9$. They are shown in fig. 2.9.

The diabatic PES derivatives are:

$$\frac{\partial H_{11}}{\partial R} = \frac{\partial H_{22}}{\partial R} = 0 \tag{2.40a}$$

$$\frac{\partial H_{12}}{\partial R} = \frac{\partial H_{21}}{\partial R} = BC \begin{cases} e^{-CR} & R \geq 0 \\ e^{CR} & R < 0 \end{cases}. \tag{2.40b}$$

They are shown in fig. 2.10.

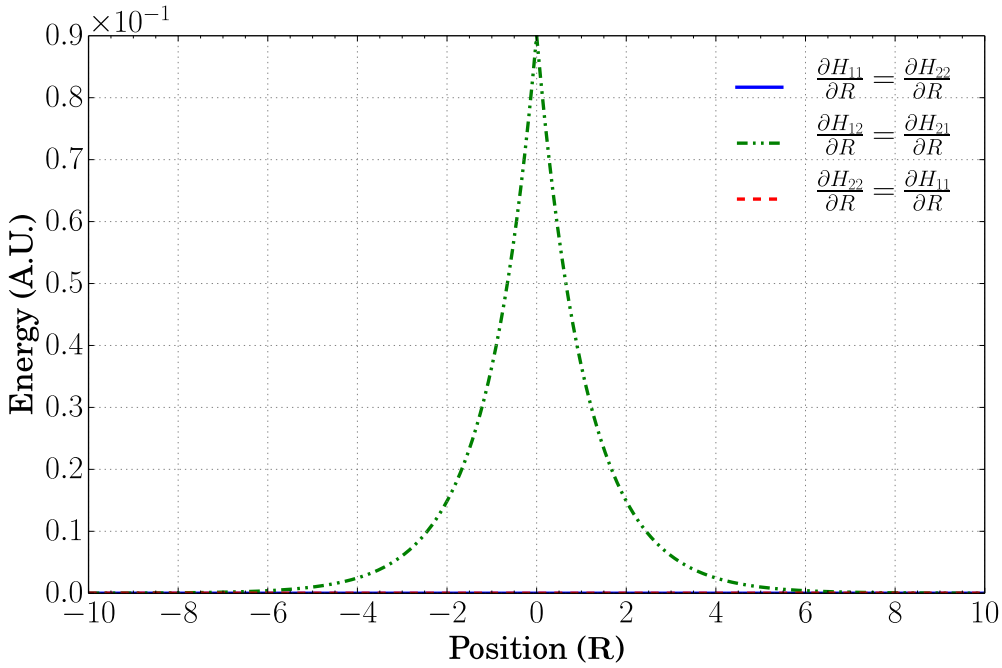


Figure 2.10: Diabatic PES derivatives.

The adiabatic PES are:

$$E_1 = -\sqrt{A^2 + B^2 \begin{cases} (2 - e^{-CR})^2 & R \geq 0 \\ e^{2CR} & R < 0 \end{cases}} \tag{2.41a}$$

$$E_2 = -E_1 \ . \tag{2.41b}$$

They are shown in fig. 2.11.

Their derivatives are:

$$\frac{\partial E_1}{\partial R} = \begin{cases} -\dfrac{B^2 C e^{-2CR}\left(2e^{CR} - 1\right)}{\sqrt{A^2 + B^2\left(e^{-CR} - 2\right)^2}} & R \geq 0 \\ -\dfrac{B^2 C e^{2CR}}{\sqrt{A^2 + B^2 e^{2CR}}} & R < 0 \end{cases} \tag{2.42a}$$

$$\frac{\partial E_2}{\partial R} = -\frac{\partial E_1}{\partial R} \ . \tag{2.42b}$$

They are shown in fig. 2.12.

The energy difference between both adiabatic PES is shown in fig. 2.13.

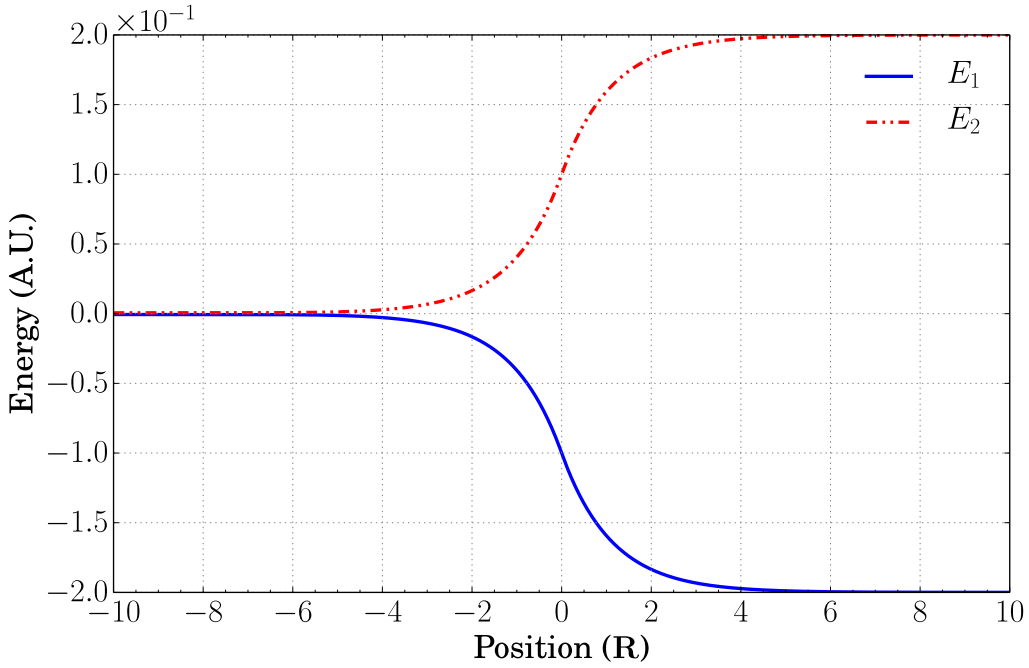


Figure 2.11: Adiabatic PES. Eigenvalues of the diabatic Hamiltonian matrix.

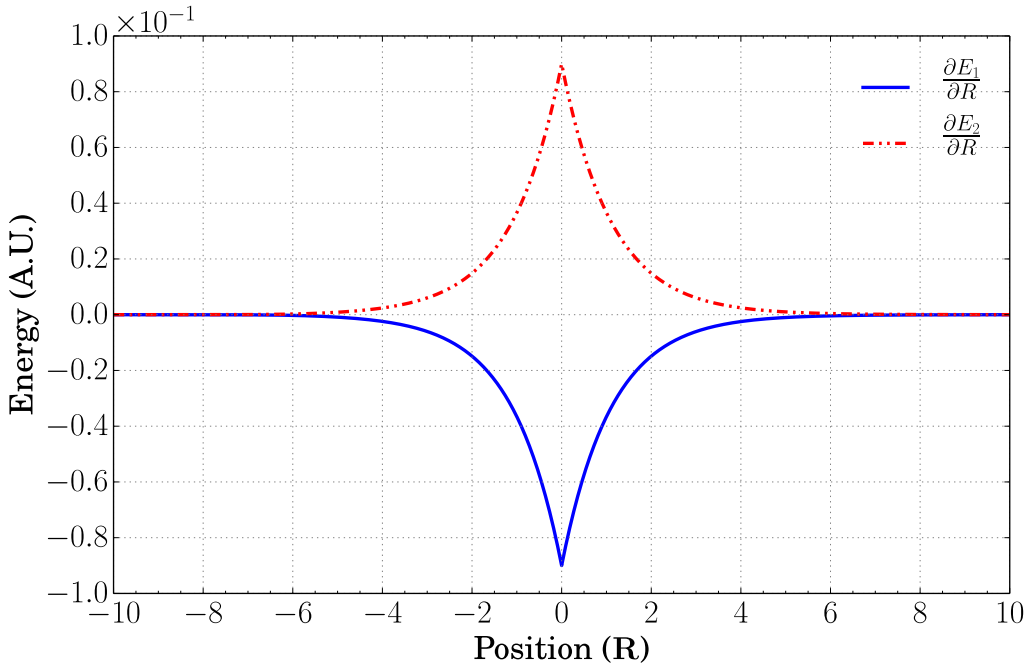


Figure 2.12: Adiabatic PES derivatives. Derivatives of the eigenvalues of the diabatic Hamiltonian matrix.

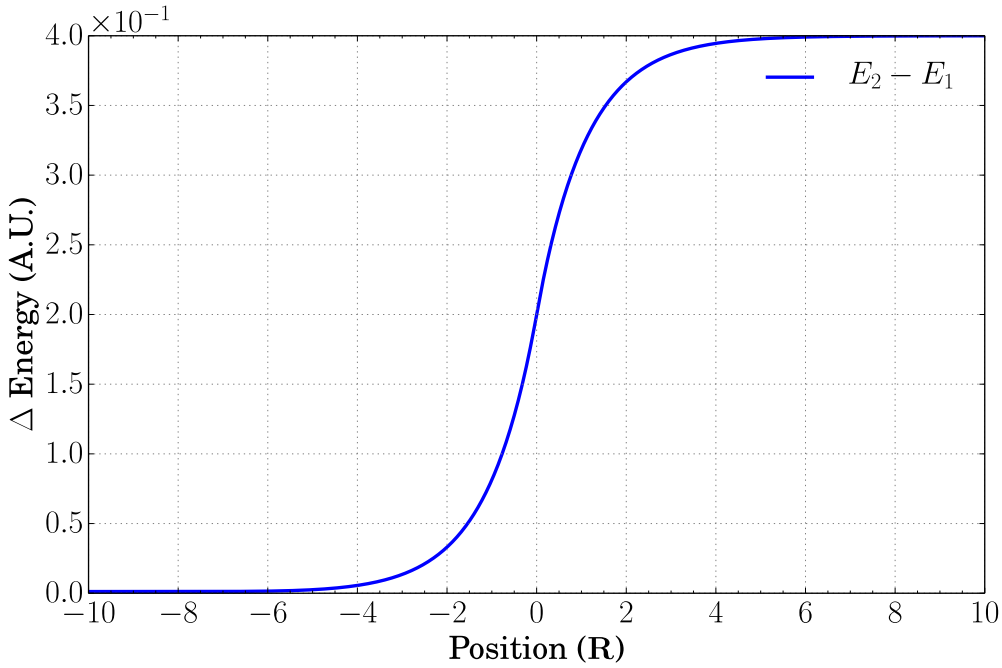


Figure 2.13: Energy difference between adiabatic PES.

Equations of Motion

The equations of motion are:

$$\dot{R} = \frac{\chi}{\mu} \tag{2.43a}$$

$$\dot{P} = \frac{ABC}{2}\left(\begin{cases} e^{-CR} & R \geq 0 \\ e^{CR} & R < 0 \end{cases}\right)\left(\frac{(p_1^2 + x_1^2 - p_2^2 - x_2^2)\cdot\phi}{\eta} - \frac{(p_2 x_1 - p_1 x_2)\cdot\chi}{\mu\cdot\eta^2}\right) \tag{2.43b}$$

$$\dot{x_1} = -\zeta\cdot x_2 - \eta\cdot p_1 \tag{2.43c}$$

$$\dot{p_1} = -\zeta\cdot p_2 + \eta\cdot x_1 \tag{2.43d}$$

$$\dot{x_2} = \zeta\cdot x_1 + \eta\cdot p_2 \tag{2.43e}$$

$$\dot{p_2} = \zeta\cdot p_1 - \eta\cdot x_2\ , \tag{2.43f}$$

where,

$$\phi = \frac{B\begin{cases}(2 - e^{-CR}) & R \geq 0 \\ e^{CR} & R < 0\end{cases}}{A} \tag{2.44a}$$

$$\eta = A\sqrt{1 + \phi^2} \tag{2.44b}$$

$$\chi = P + \frac{1}{2}(p_2 x_1 - p_1 x_2)\arctan(\phi) \tag{2.44c}$$

$$\zeta = \frac{\arctan(\phi) \cdot \chi}{2\mu} \ . \tag{2.44d}$$

2.4.4 Spin-Boson Model for Condensed-Phase Dynamics

This problem is vastly different from the others. In our case, the model describes a 1D system of M coupled oscillators. More specifically, a one-dimensional lattice made up of M oscillating nuclei which posses bulk electronic states. What is measured here are not trajectories, but rather the time-dependent difference in the probability of finding the system in either of two electronic states; when the initial state = 1, $D(t) = P_{1\leftarrow 1} - P_{2\leftarrow 1}$, when the initial state = 2, $D(t) = P_{1\leftarrow 2} - P_{2\leftarrow 2}$.

Potential Energy Surfaces

The diabatic PES are defined as follows:

$$H_{11}(\boldsymbol{Q}) = V_0(\boldsymbol{Q}) + V_1(\boldsymbol{Q}) + \epsilon \tag{2.45a}$$

$$H_{22}(\boldsymbol{Q}) = V_0(\boldsymbol{Q}) - V_1(\boldsymbol{Q}) - \epsilon \tag{2.45b}$$

$$H_{12}(\boldsymbol{Q}) = H_{12}(\boldsymbol{Q}) = \Delta \ , \tag{2.45c}$$

where,

$$V_0(\boldsymbol{Q}) = \sum_{k=1}^{M} \frac{1}{2} m_k \omega_k^2 Q_k^2 \tag{2.46a}$$

$$V_1(\boldsymbol{Q}) = \sum_{k=1}^{M} c_k Q_k \ . \tag{2.46b}$$

For our purposes, the frequencies, ω_k, are uniformly distributed in the interval $[0.01\omega_c,\ 4\omega_c]$ [3], where ω_c is known as the *characteristic frequency*, and defines the system's overall time scale. The observant reader will realise this does not make a lot of sense on its own, because not all frequencies contribute to a system's energy in equal amounts. Which is why the coupling parameters, c_k, are chosen so they obey the distribution:

$$J(\omega) = \frac{\pi}{2} \sum_{k=1}^{M} \frac{c_k^2}{m_k \omega_k} \delta(\omega - \omega_k) \ , \tag{2.47}$$

where $J(\omega)$ is an Ohmic distribution defined as:

$$J(\omega) = \frac{\pi}{2}\alpha\omega \exp\left(-\frac{\omega}{\omega_c}\right) \ , \tag{2.48}$$

where α is the Kondo parameter (a coupling strength parameter). After equating eqs. (2.47) and (2.48), and integrating with respect to ω, we find the values of c_k to be:

$$c_k = \sqrt{\frac{2}{\pi}\Delta\omega m_k \omega_k J(\omega_k)} = \omega_k \sqrt{\alpha \Delta\omega m_k \exp\left(-\frac{\omega_k}{\omega_c}\right)} \tag{2.49a}$$

$$\Delta\omega = \frac{\omega_{max} - \omega_{min}}{M-1} \ . \tag{2.49b}$$

The initial electronic conditions are initiated in the same manner as before. However, the nuclear ones are not. As stated before, the model describes oscillating nuclei in a 1D lattice, and their momenta and positions are assumed to follow the Wigner distribution:

$$\rho(\boldsymbol{P}, \boldsymbol{Q}) = \prod_{k=1}^{M} \exp\left(-a_k \frac{P_k^2}{2m_k}\right) \exp\left(-a_k \frac{1}{2} m_k \omega_k^2 \left[Q_k + \frac{c_k}{m_k \omega_k^2}\right]^2\right) \tag{2.50a}$$

$$a_k = \frac{2}{\omega_k} \tanh\left(\frac{\beta\omega_k}{2}\right) \ . \tag{2.50b}$$

Which, in reality, is a bivariate normal distribution. Fortunately, the lack of a covariant term implies that positions and momenta can be independently sampled from normally distributed random numbers. In other words, the nuclei follow independent normal distributions for their momenta and positions. It is also worth noting that the argument for the total distribution is basically a scaled quasiclassical analogue of the quantum harmonic oscillator Hamiltonian operator ($\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\omega^2\hat{x}^2$), which makes a lot of intuitive sense, because it means the nuclear parameters are sampled according to the oscillators' normally distributed kinetic energy.

Sampling is done in the traditional way: assuming X is a normally distributed random number with unitary standard deviation and mean equal to zero, a normally distributed random number G with standard deviation equal to σ and mean equal to μ can be calculated by $G = \sigma X + \mu$. Given the definition of the normal distribution,

$$G(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \ , \tag{2.51}$$

the standard deviations and mean values of the nuclei's momenta and positions are:

$$\sigma_{P_k} = \sqrt{\frac{m_k\omega_k}{2\tanh\left(\frac{\beta\omega_k}{2}\right)}} \tag{2.52a}$$

$$\mu_{P_k} = 0 \tag{2.52b}$$

$$\sigma_{Q_k} = \sqrt{\frac{1}{2m_k\omega_k\tanh\left(\frac{\beta\omega_k}{2}\right)}} \tag{2.52c}$$

$$\mu_{Q_k} = -\frac{c_k}{m_k\omega_k^2}\ . \tag{2.52d}$$

The Gaussian sampling algorithm used, is a modified version of the Ziggurat algorithm originally written by George Marsaglia and Wai Wan Tsang[5]. The modification randomises the seed every time the subroutine is initialised, thus providing the accurate pseudo-random numbers necessary for reliable Monte Carlo simulations.

Equations of Motion

The equations of motion are obtained in exactly the same way as before (using the diabatic Hamiltonian), only we have defined $M = 100$ so there are 100 nuclear degrees of freedom (DOFs), which means there are 100 equations of motion for position and another 100 for momentum, giving a grand total of 204 equations. The nuclear equations of motion—as well as H_{11} and H_{22}—are implemented as `do` loops:

$$\dot{Q}_j = \frac{P_j}{m_j} \tag{2.53a}$$

$$\dot{P}_j = -m_j\omega_j^2 Q_j - \frac{1}{2}c_j\cdot(p_1^2 + x_1^2 - p_1^2 - x_2^2) \tag{2.53b}$$

$$\dot{x_1} = p_1(V_1 + \epsilon) + p_2\Delta \tag{2.53c}$$

$$\dot{p_1} = -x_1(V_1 + \epsilon) - x_2\Delta \tag{2.53d}$$

$$\dot{x_2} = -p_2(V_1 + \epsilon) + p_1\Delta \tag{2.53e}$$

$$\dot{p_2} = x_2(V_1 + \epsilon) - x_1\Delta\ . \tag{2.53f}$$

2.5 Code Structure

- Main program.
  1. Call data collection subroutine.
  2. Call print PES and DPES subroutine.
- Data collection subroutine.

[5] See http://people.sc.fsu.edu/~jburkardt/f_src/ziggurat/ziggurat.html.

1. Declare and initialise variables.
2. Open output file.
3. Write heading on file.
4. Define value of initial momentum.
5. Call Monte-Carlo averaging routine.
6. Write final results.
   - Repeat 4. to 6. for other initial momentum values.

- Monte-Carlo averaging subroutine.

1. Declare and intialise variables.
2. Define Monte-Carlo steps loop.
   (a) Call solution subroutine.
   (b) Add results.
3. Average results with the number of Monte-Carlo steps.
4. Calculate transition probabilities for reflection and transmission.

- Solution subroutine.

1. Declare and initialise variables.
2. Define and calculate initial conditions and window functions.
3. Define integration loop.
   (a) Call RK4G subroutine using the appropriate equations of motion as argument.
   (b) Check for reflection, exit loop if the particle was reflected.
   (c) Update the initial conditions for next integration step.
4. Calculate final values for the acceptance criteria.
5. Calculate final window functions.
6. Calculate results for the Monte-Carlo averaging subroutine.

- Equations of motion subroutine.

1. Define and initialise variables.
2. Define equations of motion.

- Runge-Kutta 4 Gill numerical integration routine (RK4G).

1. Define and initialise variables.

2. Call equations of motion subroutine.
   - Calculate the intermediate step for all independent variables.
3. Repeat 2. for every intermediate step.
4. Calculate the final value of every independent variable, to be used in the next integration step.

- PES and DPES printing subroutine.
  1. Define and initialise variables.
  2. Define loop to print all functions.
     (a) Open relevant output file.
     (b) Call PES and DPES subroutines.
     (c) Write numerical values of each PES and DPES function, evaluated at any given distance.
         - Repeat b) and c) for all desired distances.

- PES or DPES subroutines.
  1. Define and initialise variables.
  2. Check which PES or DPES are required.
  3. Define PES or DPES.

Chapter 3

Results

The code was found to scale linearly with integration step, number of Monte-Carlo repetitions (MC reps) and number of cores used—which are the best case scenarios; this is what everyone who does programming hopes for. There were some minor variations from exact linearity but their randomness is due to the model's Monte-Carlo characteristics. It was also found that accuracy is more dependent on the number of MC reps than the size of the integration step, h. In fact, the integration step could be made relatively large (how large depends on the problem) without significantly affecting accuracy, but drastically reducing execution time. There was also no significant difference between using $\gamma = \dfrac{\sqrt{3}-1}{2}$ and $\gamma = 0.366$ [1]—the actual value is very similar, but there is no need to evaluate square roots and divisions all the time if we use the numerical approximation; [1] gives the theoretical justification for this value, but it can be adjusted on a case by case basis. Accuracy was also not noticeably affected by parallel runs, presumably due to the random number generator used—which is thread safe—but not optimal for parallelisation, because it generates numbers in sequence from a single source rather than in parallel[1] (this only affects execution speed).

The number of MC reps = 15000 and the value of $\gamma = 0.366$ unless stated otherwise. Reflections are denoted as R and transmissions as T, electronic transitions from an initial state, i, to a final state, f, are denoted as $f \leftarrow i$.

All calculations were carried out on a computer equipped with an Intel i7-3770k processor overclocked to 4.3 GHz, 2×4 Gb Kingston RAM overclocked to 1660 MHz, mounted on a Gigabyte GA-Z77X-UD3H motherboard (the video card is irrelevant because all calculations were processor bound), all functioning on the `ubuntu 14.04 LTS` operating system. The code was compiled with the `gfortran` module from the `GNU` compiler collection `gcc 4.9.2` [2].

[1] See https://gcc.gnu.org/onlinedocs/gfortran/RANDOM_005fNUMBER.html
[2] See https://gcc.gnu.org/

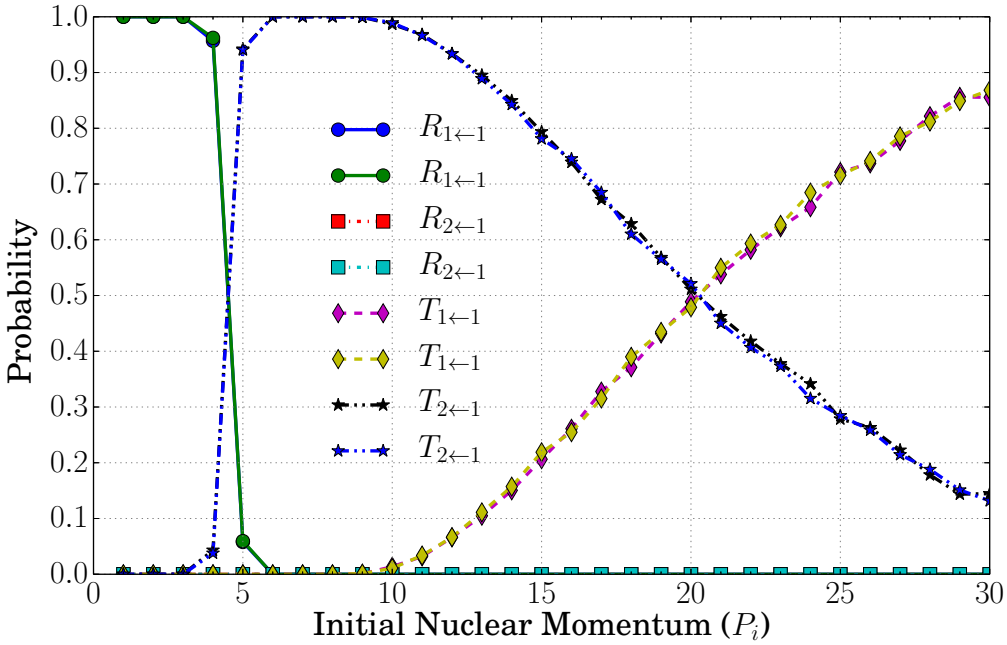


Figure 3.1: Comparison between $h = (5P_i)^{-1}$ (appearing first in the legend) and $h = P_i^{-1}$. $\gamma = \frac{\sqrt{3}-1}{2}$ in both cases. $i = 1$.

3.1 Single Avoided Crossing

Being the simplest system, the single avoided crossing problem was used to test code's robustness. Figures 3.1 to 3.4 show results for $i = 1$ while figs. 3.5 to 3.6 show the results for $i = 2$. Despite this being the simplest problem tackled herein, it presents some fairly interesting behaviours. The results for when $i = 1$ are very similar—though not exactly the same—as those found in [1]. However, this is most likely down to the fact that they used 50000–100000 trajectories as well as there being a strong possibility that the compiler, RNG, and integration step differ.

Figure 3.1 shows a comparison between two different integration step sizes, there is no significant difference in accuracy between them both. Figure 3.2 shows the results for very large values of h, there is no significant difference between using large or small values of h. Figure 3.3 shows a two-core parallel run, there is no significant difference in accuracy compared to single-core calculations.

First, we shall tackle the case where $i = 1$. The behaviour of $R_{1\leftarrow1}$, whose probability is seen to rapidly decline as nuclear momentum increases, can be explained by referring back to fig. 2.1. For small values of nuclear momentum, there is not enough kinetic energy to scale over the potential barrier (akin to an 'activation energy') and continue in the original direction, thus the particle is reflected back, preventing its electronic states from interacting, as seen in figs. 3.4a and 3.4b. $R_{2\leftarrow1}$ is similarly

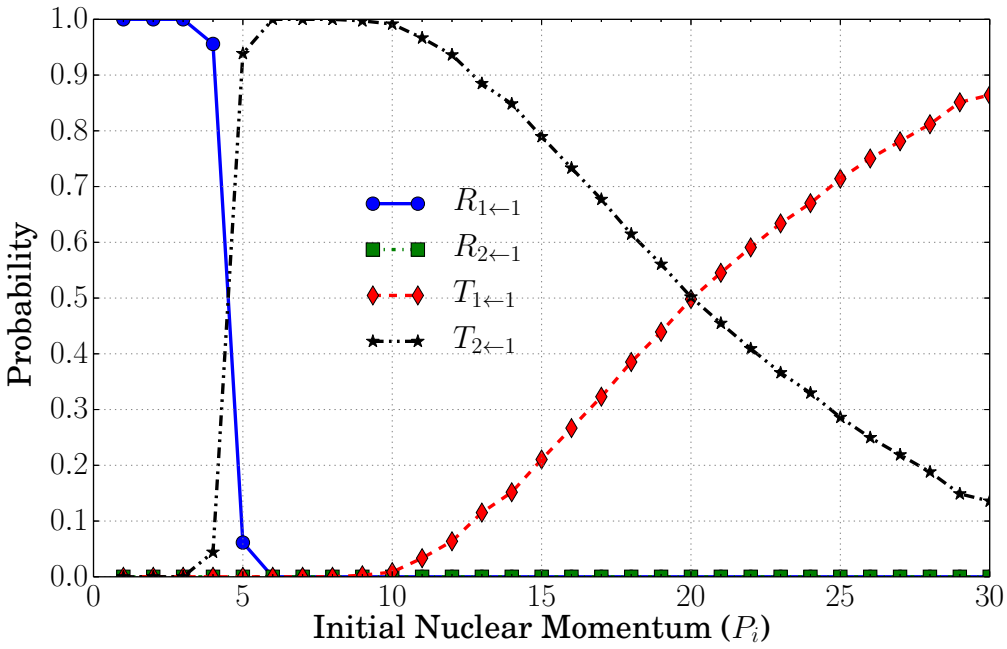


Figure 3.2: Transition probabilities for R and T, $h = (0.012P_i)^{-1}$. $i = 1$. Computational time = 8 min. 7.35 s.

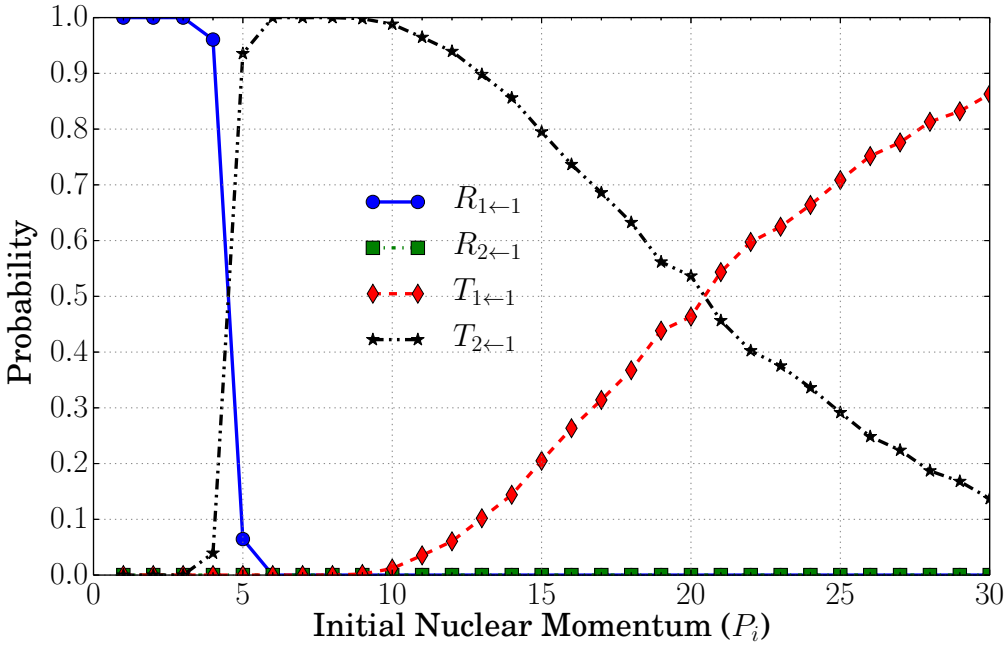
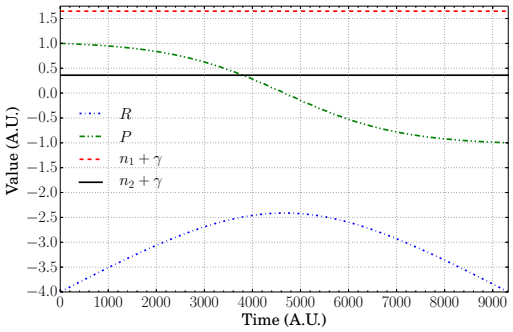


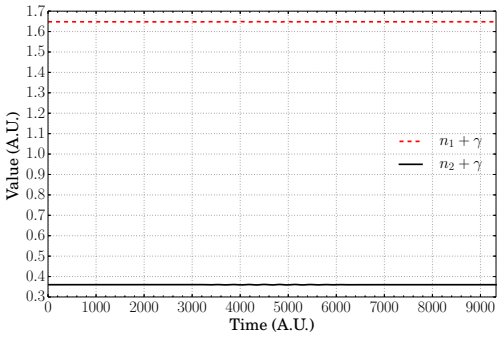
Figure 3.3: Parallel calculations on two cores (7500 MC reps per core), $h = (5P_i)^{-1}$. No significant deviation from single-core calculations. $i = 1$.

explained; once there is enough kinetic energy to reach a point where a transition is likely to happen, there would also be enough energy to continue on to the other side. Thus $R_{2\leftarrow 1}$ remains relatively close to zero throughout. The others can be explained by the fact that transition probabilities are not only a function of the distance between adiabatic PES, but also of the time it takes the particle to move away from regions where electronic coupling is strong. This explains the behaviour of $T_{2\leftarrow 1}$ and $T_{1\leftarrow 1}$. For $T_{2\leftarrow 1}$ lower nuclear momenta allow more time for an electronic transition to occur as seen in figs. 3.4c and 3.4d; however, after a certain value of nuclear momentum, it begins to decrease in favour of $T_{1\leftarrow 1}$ because the particle travels so fast, it cannot interact long enough to undergo an electronic transition, as shown in figs. 3.4d and 3.4e.

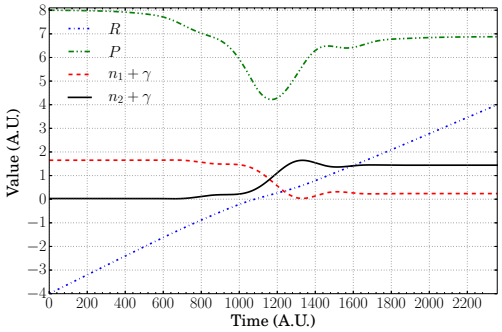
In the case of $i = 2$ only one graph is provided (fig. 3.5), as tests yielded similar results as previously demonstrated. In this case, the system's behaviour is more nuanced, but is explained by the same reasons. The $R_{1\leftarrow 2}$ transition remains effectively zero throughout, because at such low nuclear momenta two things can happen: the particle can be reflected back along H_{22} by the off-diagonal PES—corresponding to the $R_{2\leftarrow 2}$ transition, as seen in figs. 3.6a and 3.6b (in fact the particle keeps getting bounced back time and time again until it finally leaves the integration area as a reflection)—or it moves down to the lower energy state, where the surplus potential energy is converted into kinetic energy (conservation of energy) and a $T_{1\leftarrow 2}$ transition is observed, as seen in figs. 3.6c and 3.6d. For higher values of nuclear momentum, the particle has enough energy to be transmitted, but there is not enough time for the particle's electronic states to interact, and a $T_{2\leftarrow 2}$ is observed instead, shown in figs. 3.6e and 3.6f. The higher the nuclear momentum the less time there is and the likelier this transition becomes.



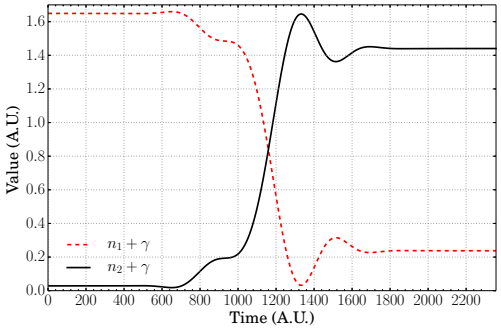
(a) $R_{1\leftarrow 1}$ trajectory.



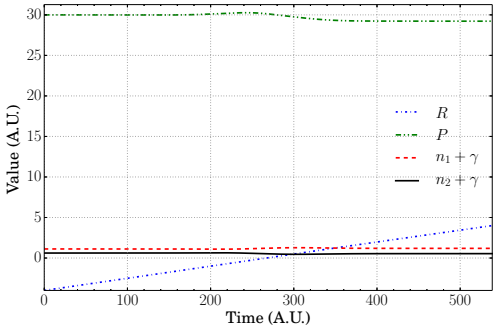
(b) $R_{1\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



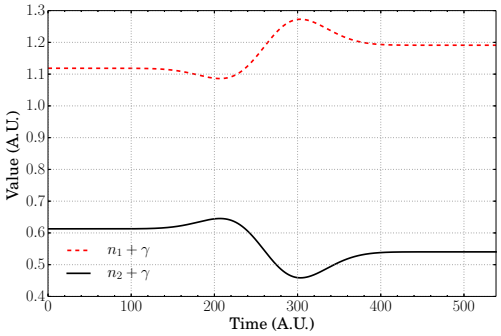
(c) $T_{2\leftarrow 1}$ trajectory.



(d) $T_{2\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



(e) $T_{1\leftarrow 1}$ trajectory.



(f) $T_{1\leftarrow 1}$ trajectory, zoom into n_1 and n_2.

Figure 3.4: Trajectory examples. $i = 1$.

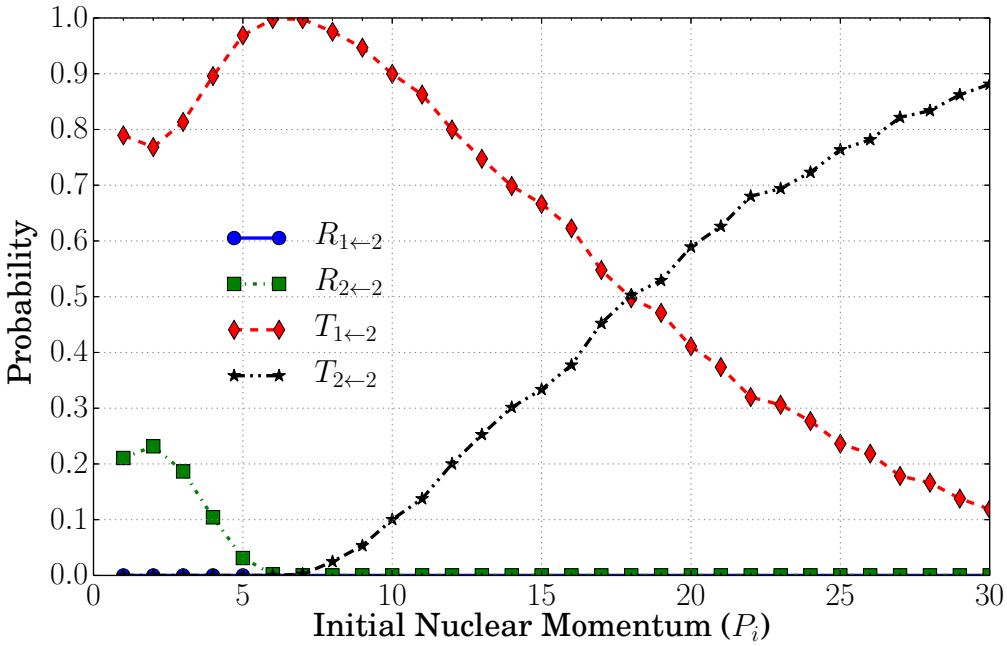
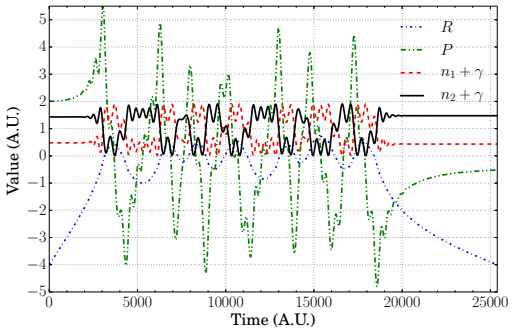
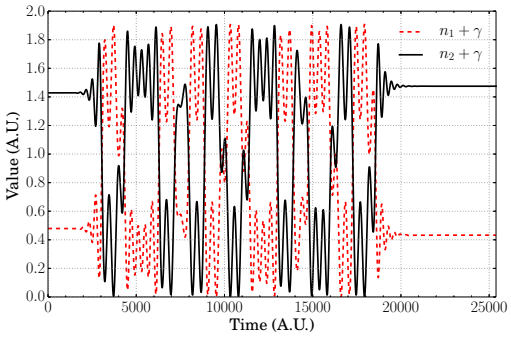


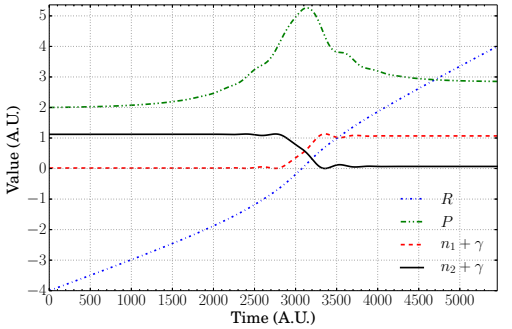
Figure 3.5: Transition probabilities for R and T, $h = (0.012P_i)^{-1}$. $i = 2$. Computational time = 7 min. 47.74 s.



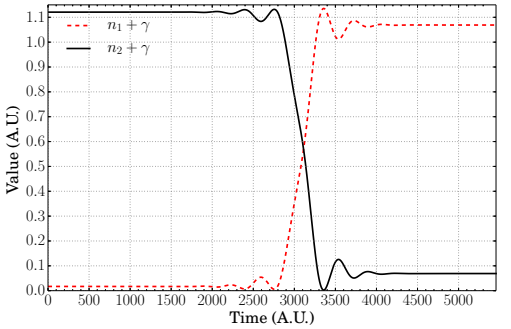
(a) $R_{2\leftarrow 2}$ trajectory.



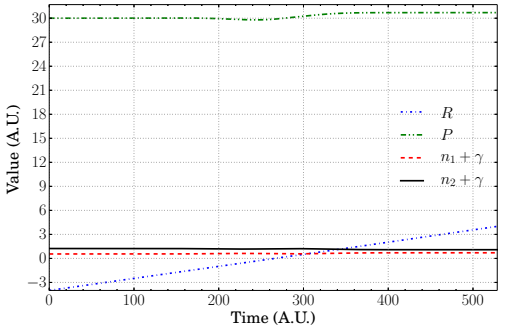
(b) $R_{2\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



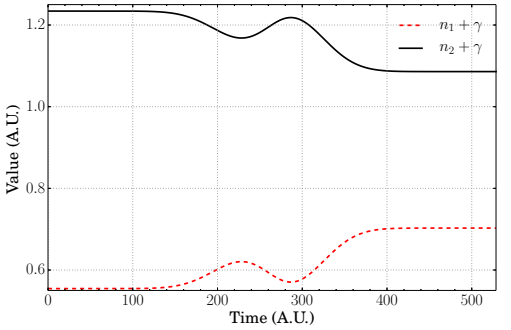
(c) $T_{1\leftarrow 2}$ trajectory.



(d) $T_{1\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



(e) $T_{2\leftarrow 2}$ trajectory.



(f) $T_{2\leftarrow 2}$ trajectory, zoom into n_1 and n_2.

Figure 3.6: Trajectory examples. $i = 2$.

3.2 Double Avoided Crossing

Figures 3.7 to 3.10 show results for $i = 1$ while figs. 3.11 to 3.12 show the results for $i = 2$. Again the results for when $i = 1$ are very similar—but not exactly the same—as those found in [1]. Which can be attributed to the same factors as before. In our case E_i is the mean total initial energy, which could also contribute to the differences with published results.

Figure 3.7 shows the results for small values of h, while fig. 3.8 uses large values of h. There is no significant difference between the two. Figure 3.9 shows a two-core parallel run. There is no significant difference between this and fig. 3.7.

Once more, we shall first analyse the behaviour for $i = 1$, shown in figs. 3.7 to 3.9. The lack of $R_{1\leftarrow 1}$ and $R_{2\leftarrow 1}$ transitions is easily explained by the fact there is no energy barrier in either the diagonal diabatic or adiabatic PES as seen in figs. 2.5 and 2.7. Therefore the particle only requires a small amount of momentum for the it to be transmitted. In fact, in low momentum tests (figs. 3.10e and 3.10f), the particle was accelerated by the adiabatic energy wells (potential energy was transferred into kinetic energy). The other interesting feature is the presence the Stückelberg oscillations for both $T_{1\leftarrow 1}$ and $T_{2\leftarrow 1}$, which are often attributed to quantum interference effects clearly seen in figs. 3.10b and 3.10d. In order to comprehend these one must only turn to fig. 2.8, which shows the difference between adiabatic PES. One can see that $E_2 - E_1$ varies quickly and by relatively large amounts. For small momenta, the particle generally does not have enough energy to switch electronic states, leading to the relatively flat appearance of both $T_{1\leftarrow 1}$ and $T_{2\leftarrow 1}$ from -4 to -2. However, as momentum increases, there is enough energy to make one jump but not a lot to make the second one, leading to the behaviour seen from -2 to -1. As momentum keeps increasing, there is enough energy for the second jump, and we get the behaviour we see from -1 to 0. Finally, when momentum is very large, the particle carries out one jump and does not linger long enough around the second minimum energy difference range, leading to the behaviour we see from 0 to 1. Figure 3.10 contains trajectory examples which support these allegations.

When the $i = 2$ as is the case in fig. 3.11, the behaviour is very similar but inverted with respect to $i = 1$. In other words, the transmission which preserves the initial state, $T_{2\leftarrow 2}$ in fig. 3.11, is very similar in shape to the transmission which preserves the initial state, $T_{1\leftarrow 1}$ in fig. 3.7. And the transmission which does not preserve the initial state, $T_{1\leftarrow 2}$ in fig. 3.11, is also very qualitatively similar to the transition which does not preserve the initial state, $T_{2\leftarrow 1}$ in fig. 3.7. Which means they have similar explanations; the only variation being the precise points at which these 'domains' appear. Trajectory examples are provided in fig. 3.12.

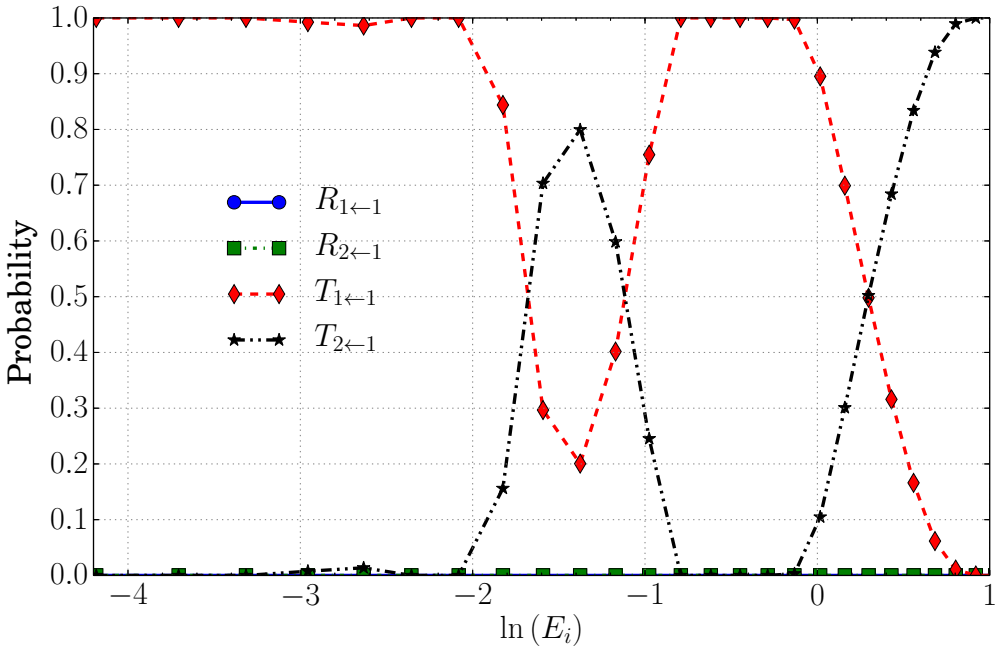


Figure 3.7: Transition probabilities for R and T, $h = (5P_i)^{-1}$. $i = 1$.

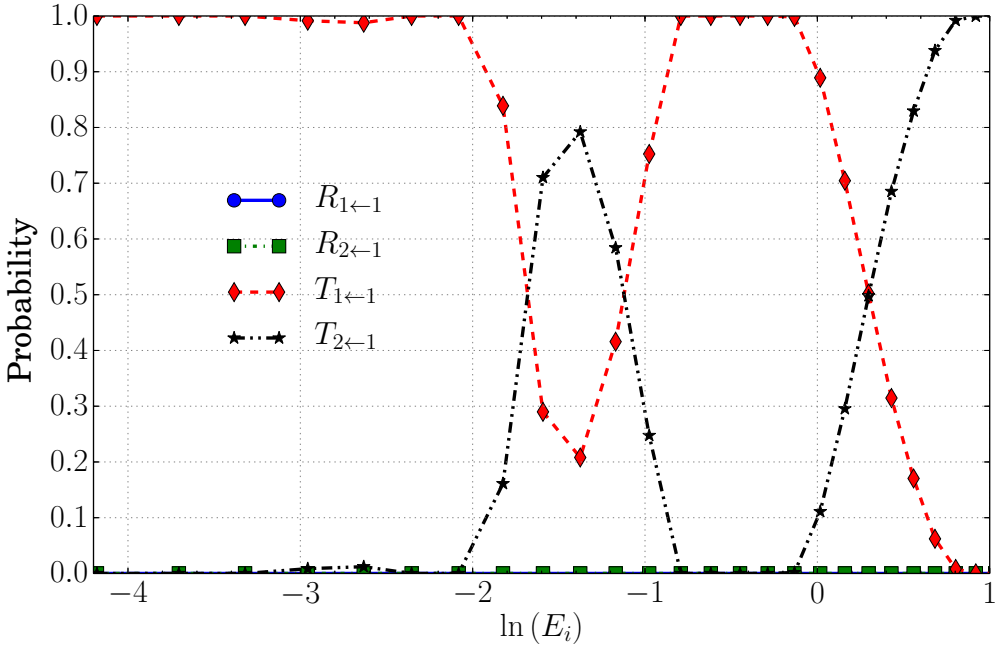


Figure 3.8: Transition probabilities for R and T for $h = (0.0125P_i)^{-1}$. $i = 1$. Computational time = 7 min. 33.29 s.

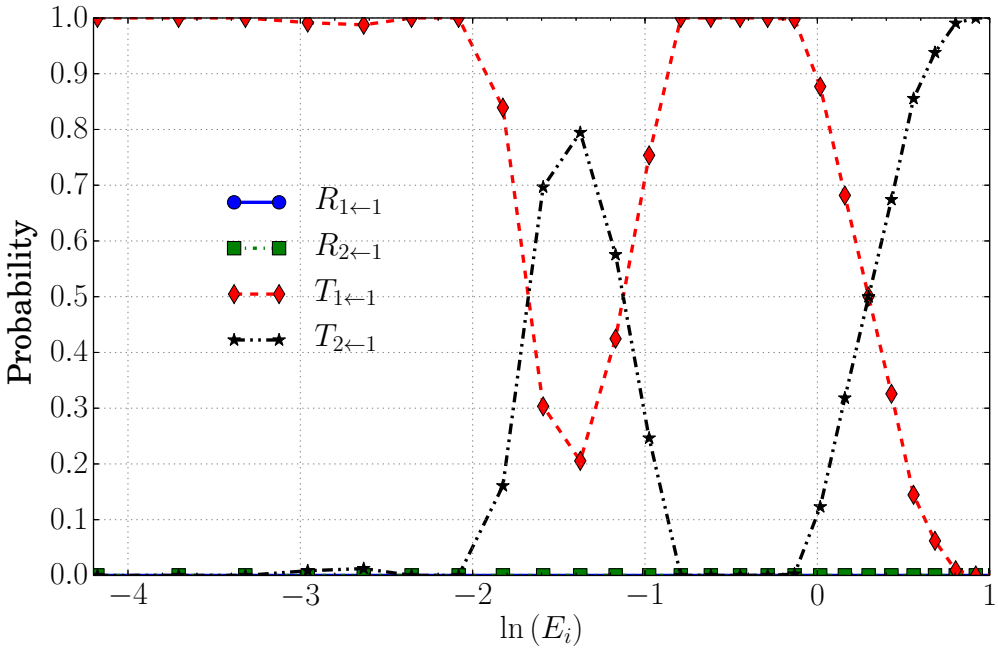
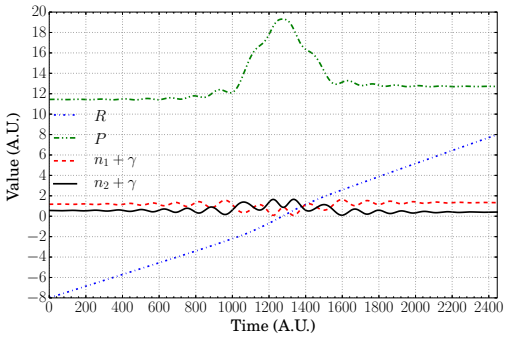
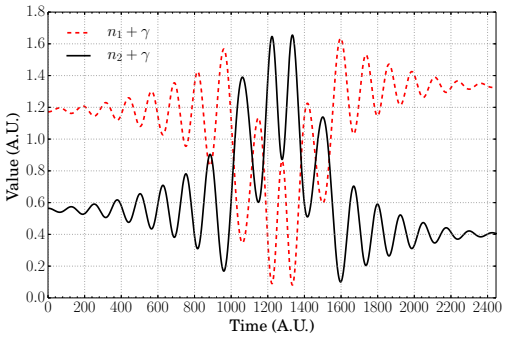


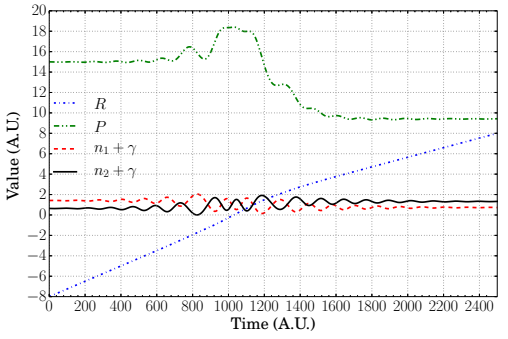
Figure 3.9: Parallel calculations on two cores (7500 MC reps per core), $h = (5P_i)^{-1}$. No significant deviation from single-core calculations. $i = 1$.



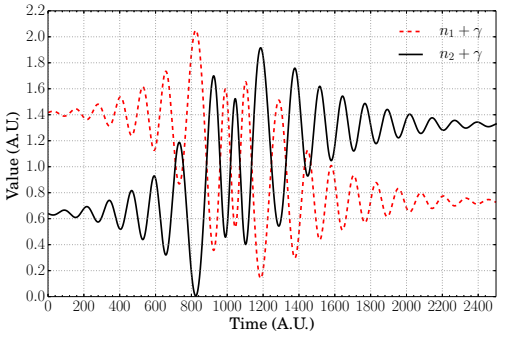
(a) $T_{1\leftarrow 1}$ trajectory.



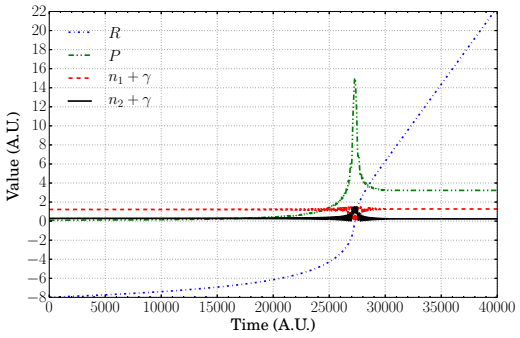
(b) $T_{1\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



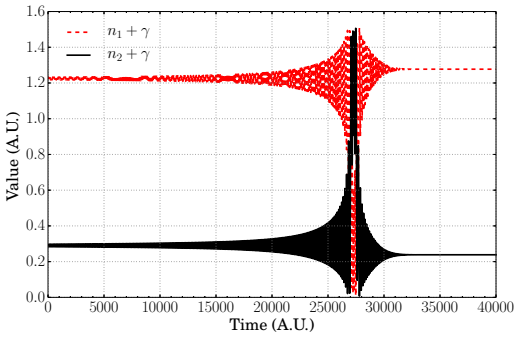
(c) $T_{2\leftarrow 1}$ trajectory.



(d) $T_{2\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



(e) Low nuclear momentum trajectory.



(f) Low nuclear momentum trajectory, zoom into n_1 and n_2.

Figure 3.10: Trajectory examples. $i = 1$.

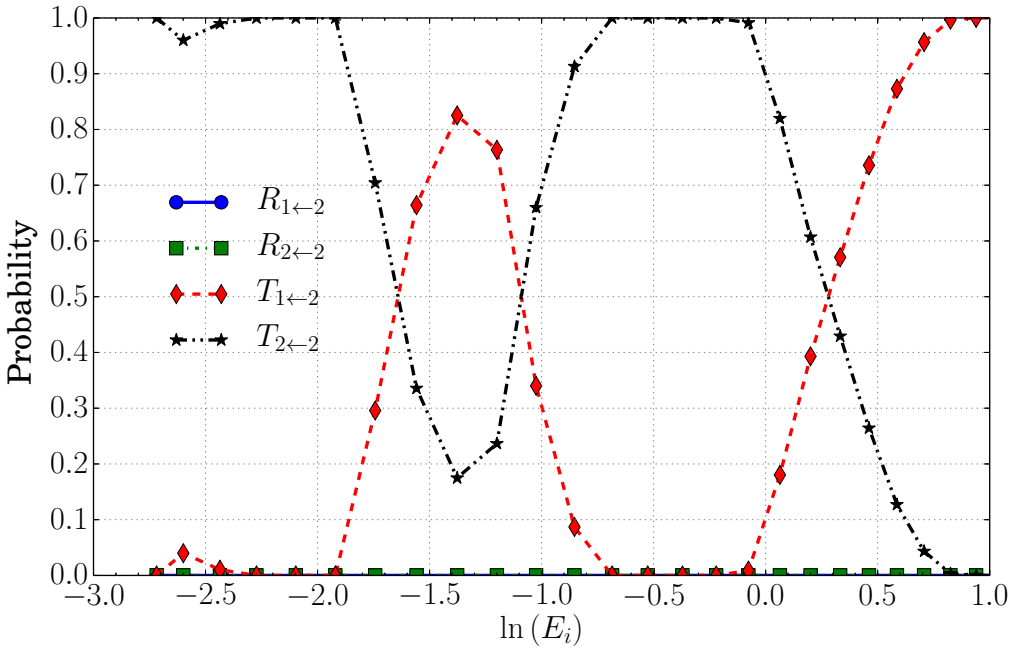
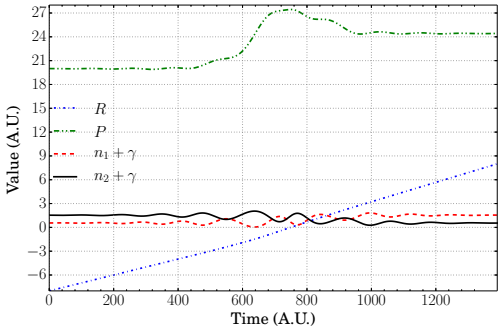
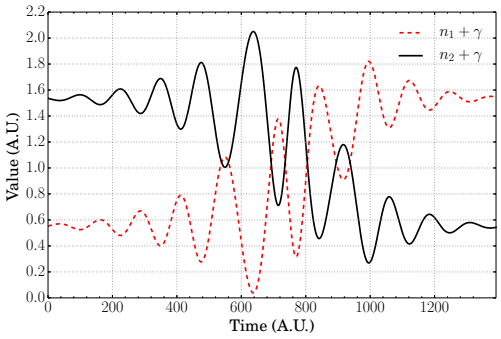


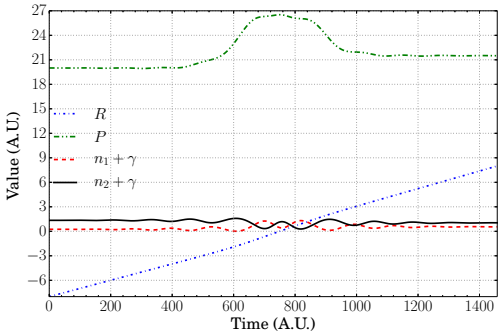
Figure 3.11: Transition probabilities for R and$T$ for $h = (0.0125P_i)^{-1}$. $i = 2$. Computational time = 10 min. 41.63 s.



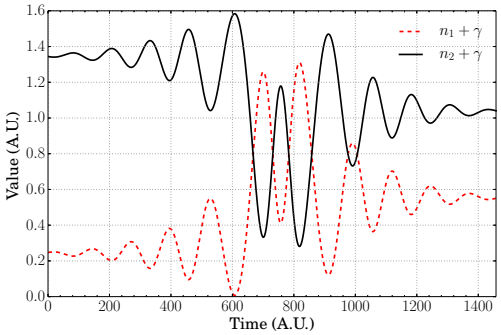
(a) $T_{1\leftarrow 2}$ trajectory.



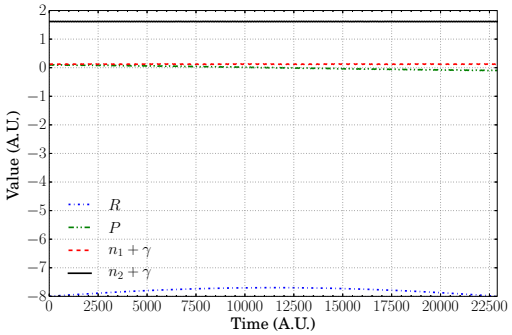
(b) $T_{1\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



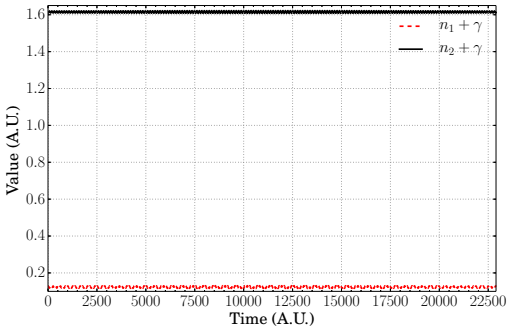
(c) $T_{2\leftarrow 2}$ trajectory.



(d) $T_{2\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



(e) Low nuclear momentum trajectory.



(f) Low nuclear momentum trajectory, zoom into n_1 and n_2.

Figure 3.12: Trajectory examples. $i = 2$.

3.3 Extended Coupling

Figures 3.13 to 3.16 show the results for $i = 1$, while figs. 3.17 to 3.18 show the results for $i = 2$. Once more, the results for when $i = 1$ are very similar—but not exactly the same—as those found in [1]. Which again can be attributed to the same factors as before.

Figure 3.13 shows the results for small values of h. They do not differ greatly from those for large values of h shown in fig. 3.14. Figure 3.15 shows results from two parallel runs, they do not differ significantly from those of single-core calculations.

Following the precedent, we shall start the analysis with $i = 1$. The behaviour of $T_{1\leftarrow 1}$ at low momenta is due to the fact that E_1 is highly attractive (see fig. 2.11), so the particle is easily transmitted without strongly interacting with any other PES, as seen in figs. 3.16e and 3.16f. However, as the momentum increases the particle starts interacting more strongly with the off-diagonal diabatic PES, and it becomes more likely to be reflected by it, with or without an electronic transition—as is observed in figs. 3.16a and 3.16c, giving rise to the bump in probability of $R_{1\leftarrow 1}$ and $R_{2\leftarrow 1}$, and the dip in the probability of $T_{1\leftarrow 1}$. It is only at higher nuclear momenta that it is possible for the particle to experience an electronic transition and have enough momentum to work against the repulsive nature of E_2—which is clearly seen in the sharp momentum drop in fig. 3.16c, which comes about by the switch in electronic state more closely observed in fig. 3.16h—leading to increased likelihood of observing $T_{2\leftarrow 1}$ transitions.

When $i = 2$ as shown in fig. 3.17, the system's behaviour changes drastically. At low nuclear momenta, the particle does not have the required energy to be transmitted or undergo an electronic transition—seen in figs. 3.18c and 3.18d—and is simply reflected by the repulsive nature of E_2, leading to the shape of $R_{2\leftarrow 2}$. However, as nuclear momentum increases, the particle starts being able to change electronic state and be transmitted or reflected as observed in figs. 3.18a, 3.18b, 3.18e and 3.18f, leading to the increase in the probabilities of $R_{1\leftarrow 2}$ and $T_{1\leftarrow 2}$. As the momentum is increased further, the probability of having enough energy to transition into the lower electronic state and be transmitted increases, leading to the drop in $R_{2\leftarrow 2}$ and $R_{1\leftarrow 2}$, and increase of $T_{1\leftarrow 2}$. As the nuclear momentum increases further, then the particle is travelling fast enough that it can overcome the repulsive effect of E_2—as observed in figs. 3.18g and 3.18h—and so it interacts relatively little with E_1, leading to the increased probability of observing $T_{2\leftarrow 2}$ transitions.

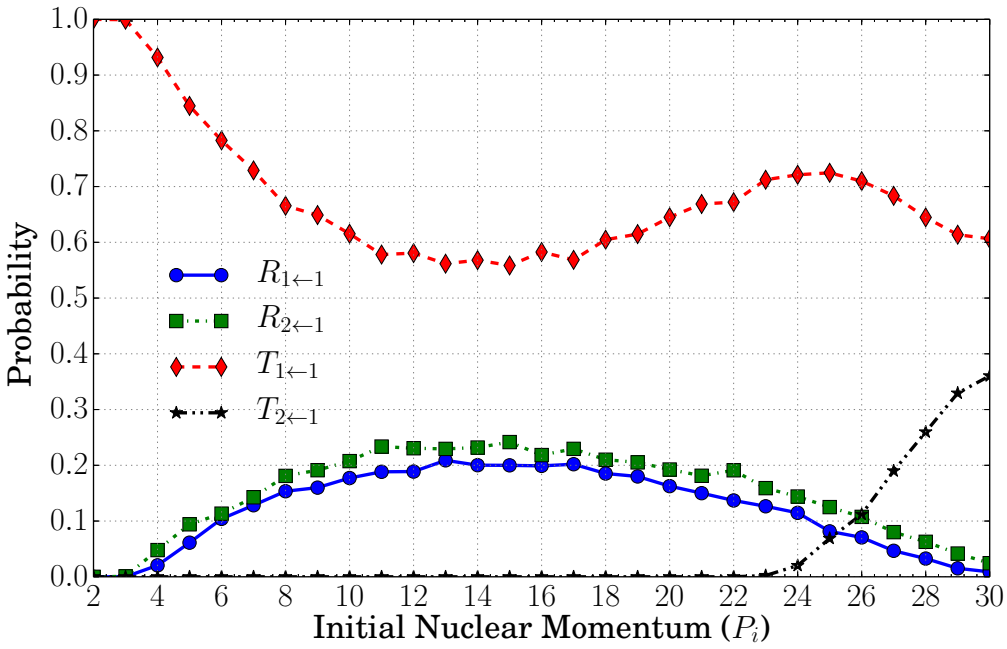


Figure 3.13: Transition probabilities for R and T for $h = (5P_i)^{-1}$. $i = 1$.

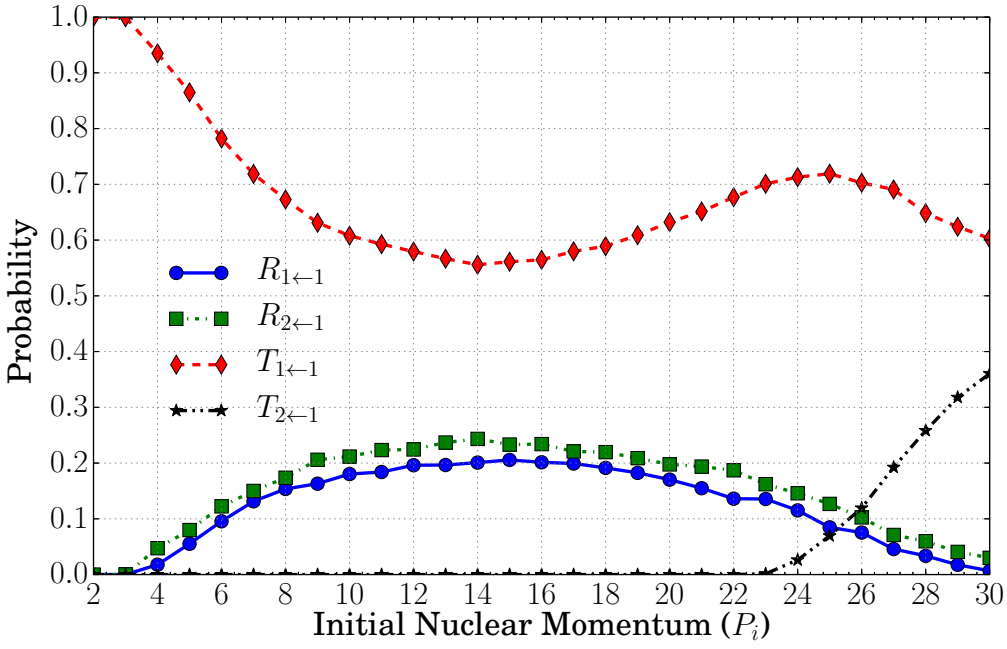


Figure 3.14: Transition probabilities for R and T for $(h = 0.10125P_i)^{-1}$. Mean of two 15,000 MC step runs, total of 30,000 MC steps. $i = 1$. Computational time $= 48 + 49$ min. $55.44 + 26.91$ s.

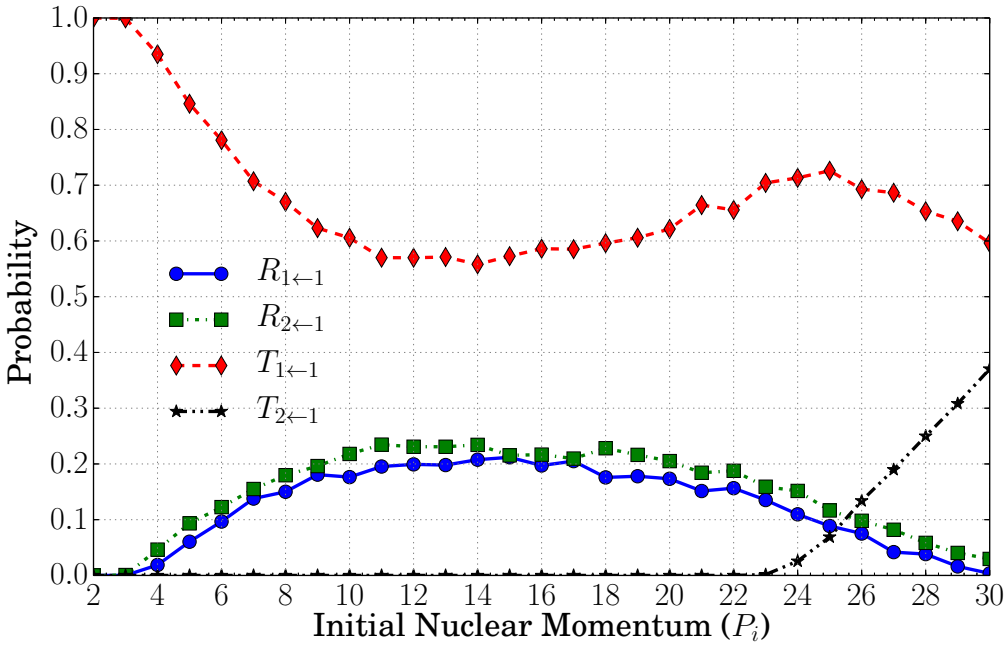
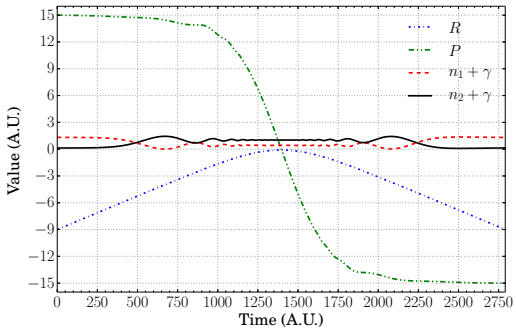
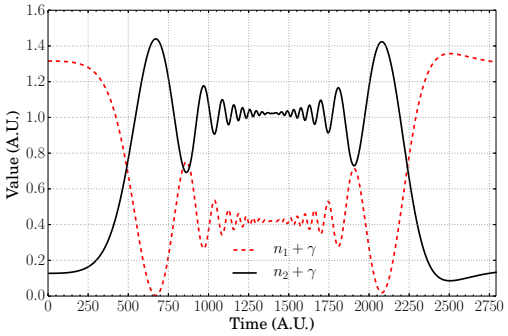


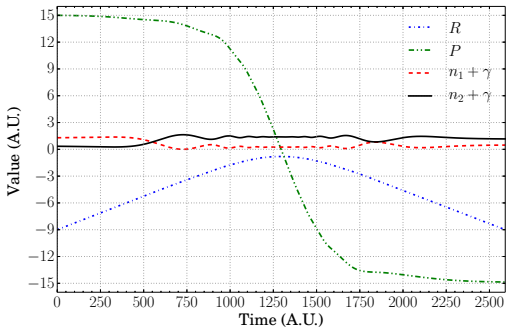
Figure 3.15: Parallel calculations on four cores (3750 per core) and $h = (5P_i)^{-1}$. $i = 1$.



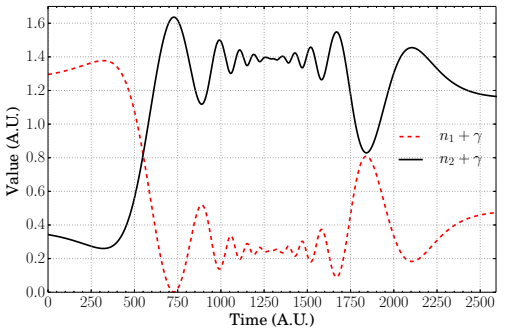
(a) $R_{1\leftarrow 1}$ trajectory.



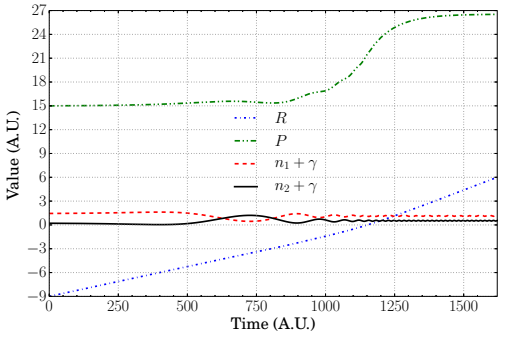
(b) $R_{1\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



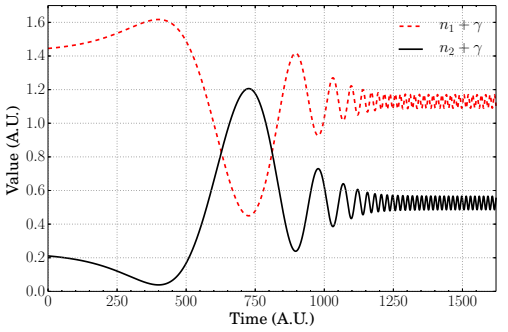
(c) $R_{2\leftarrow 1}$ trajectory.



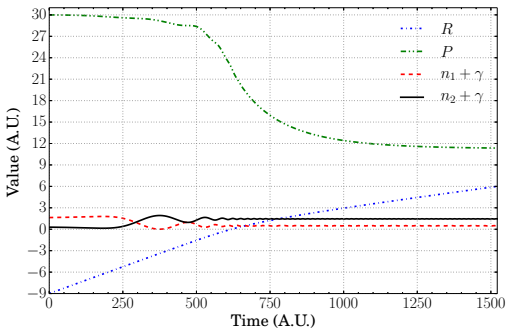
(d) $R_{2\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



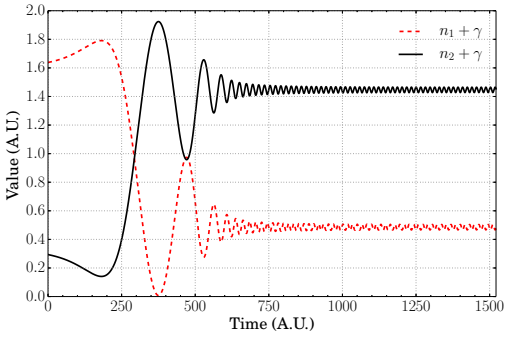
(e) $T_{1\leftarrow 1}$ trajectory.



(f) $T_{1\leftarrow 1}$ trajectory, zoom into n_1 and n_2.



(g) $T_{2\leftarrow 1}$ trajectory.



(h) $T_{2\leftarrow 1}$ trajectory, zoom into n_1 and n_2.

Figure 3.16: Trajectory examples. $i = 1$.

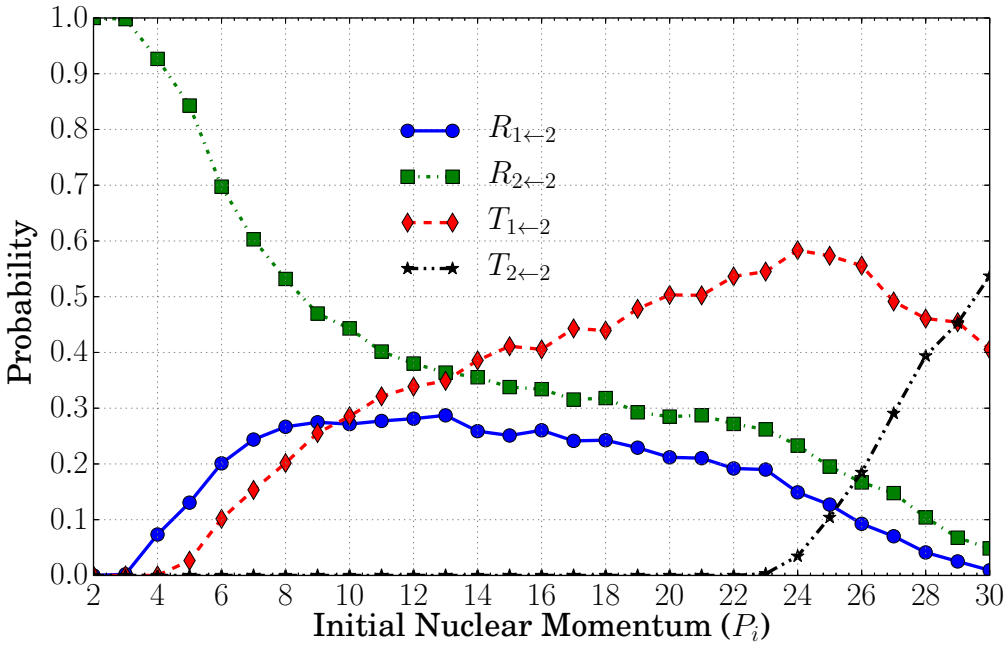
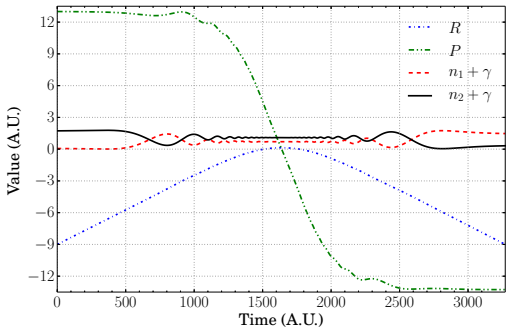
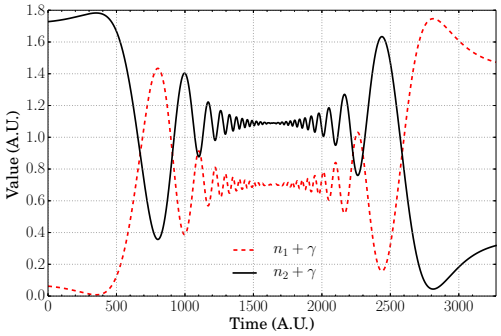


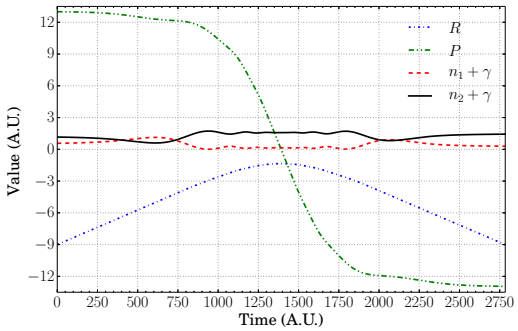
Figure 3.17: Transition probabilities for R and T for $h = (0.10125P_i)^{-1}$. $i = 2$. Computational time = 40 min. 51.63 s.



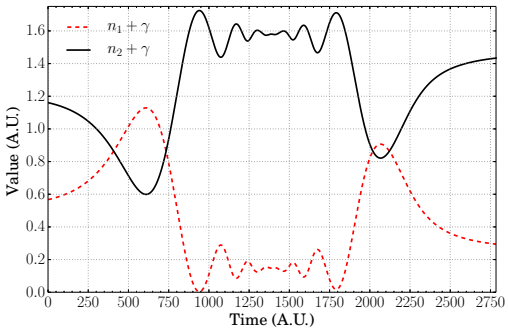
(a) $R_{1\leftarrow 2}$ trajectory.



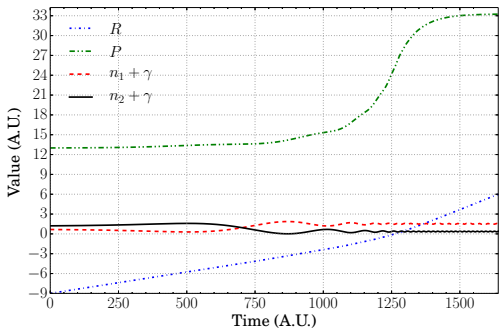
(b) $R_{1\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



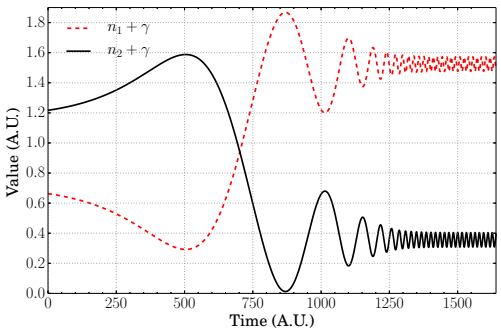
(c) $R_{2\leftarrow 2}$ trajectory.



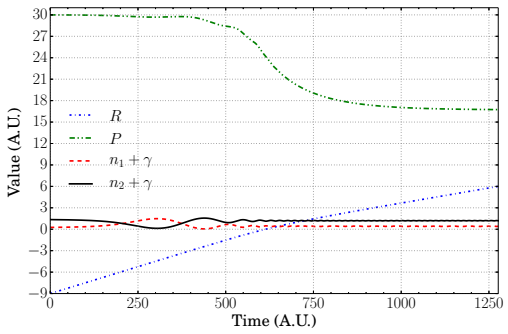
(d) $R_{2\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



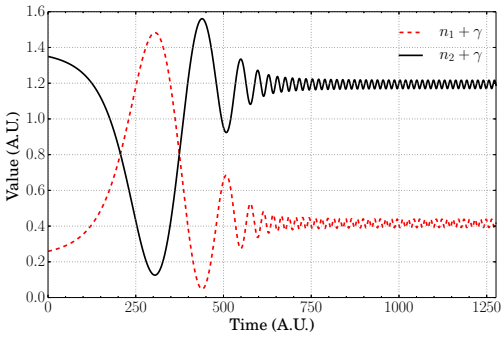
(e) $T_{1\leftarrow 2}$ trajectory.



(f) $T_{1\leftarrow 2}$ trajectory, zoom into n_1 and n_2.



(g) $T_{2\leftarrow 2}$ trajectory.



(h) $T_{2\leftarrow 2}$ trajectory, zoom into n_1 and n_2.

Figure 3.18: Trajectory examples. $i = 2$.

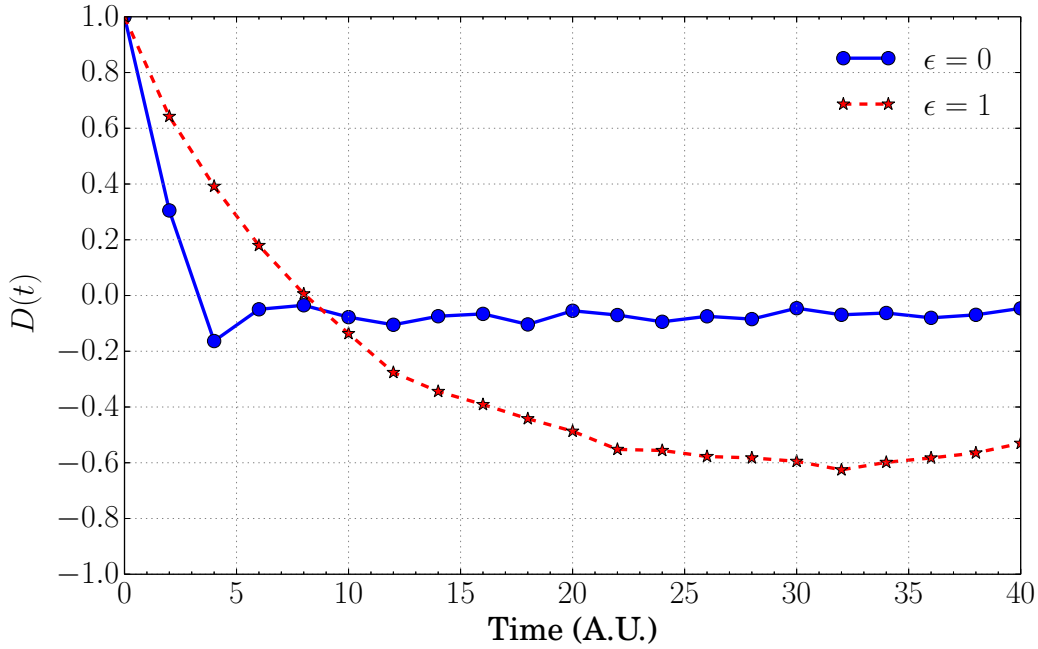
3.4 Spin-Boson Model for Condensed-Phase Dynamics

Some details were left out of [1] in regards to this problem—namely the value of ω_c, and the range and distribution of ω_k (defined in section 2.4.4)—so it was assumed that $\omega_c = 1$, and everything else was defined from there, which means there is no standard with which to compare results. Figures 3.19 and 3.20 show the results of eight systems in total: two symmetric and two asymmetric for $i = 1, 2$ each. The calculations were carried out for 100 nuclei ($M = 100$).

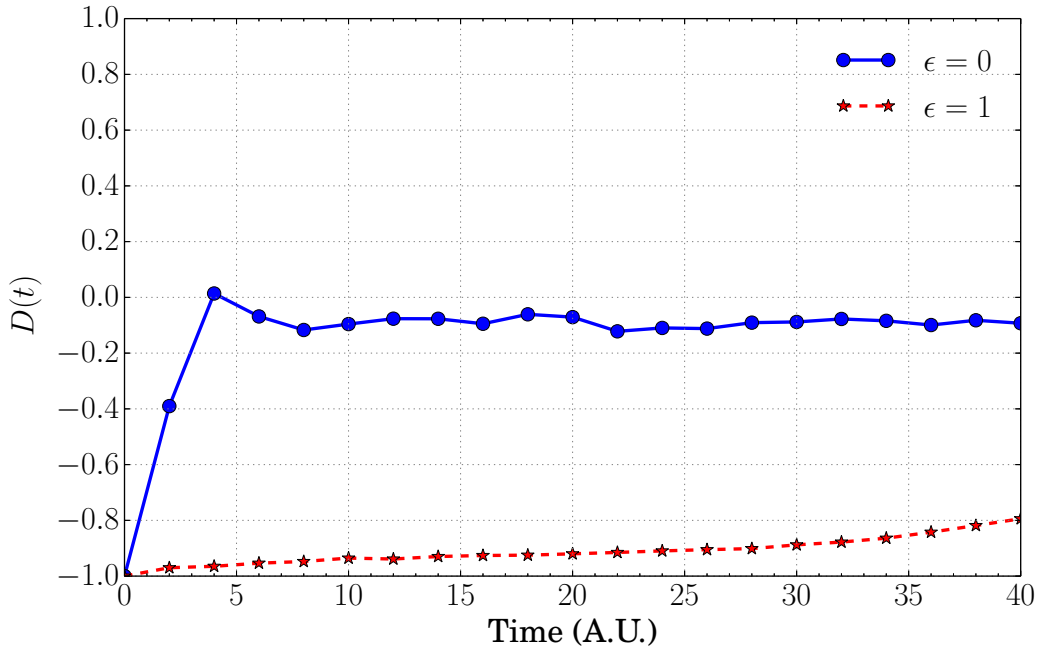
Figure 3.19 shows incoherent (non-oscillatory) relaxation dynamics for all four systems. It is particularly interesting how much the behaviour changes between the symmetric and asymmetric version of each problem. In both cases, the asymmetric system favours the second state. Which means that the second state has a lower energy than the first. This notion is also evidenced by the fact that both symmetric systems also stabilise at negative values of $D(t)$, signifying that the probability of finding the system in state 2 is higher than that of finding it in state 1. This can potentially be observed from eqs. (2.45) and (2.46).

On the other hand, fig. 3.20 shows both, coherent (oscillatory) and non-coherent (non-oscillatory) relaxation dynamics. For both initial states, the symmetric system manifests highly oscillatory behaviour. But the surprising results arise from their asymmetric counterparts. This means that for our chosen parameters, the dynamical system described by the equations of motion has large regions of stability in the electronic phase space. Such phenomena are not unheard of—where the same equations with different parameters yield vastly different results, ranging from nicely behaved to chaotic systems. Suffice to say, this was completely unexpected (and probably highly coincidental).

To conclude with this discussion, it must be said that the disparity in integration step and MC reps is down to each individual system's run time. The aforementioned quantities were adjusted so that no run exceeded 17 hours. There was some overcompensation in a few systems due to the highly variable nature of their run times, which in one instance was observed to exceed 250000%, though less extreme examples fell in the range of 300% to 1000%. This phenomenon is attributed to the large number of uniformly and normally distributed random numbers required to set the initial conditions, as some of them do not allow the selection criteria to be met. The problem was compounded by the random seed allocation at every MC rep, and by the fact that the selection criteria (eq. (2.19)) must be met at every plot point, otherwise the MC rep has to be restarted.

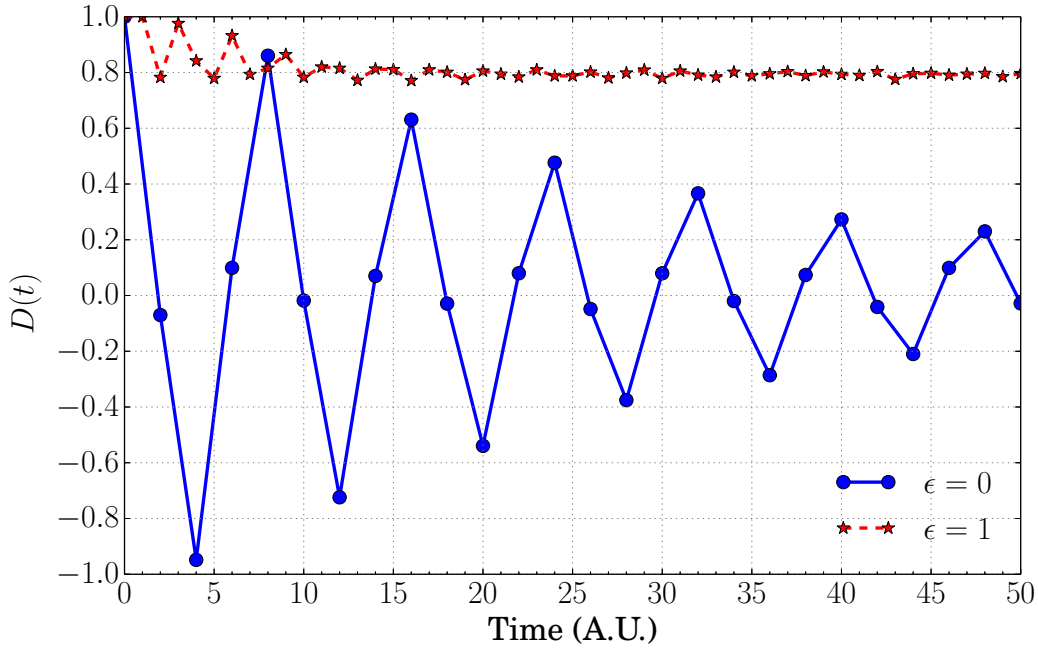


(a) $i = 1$. For both systems $h = 0.01$. Computational time symmetric: 8 hrs. 46 min. 24.33 s.; asymmetric: 3 hrs. 32 min. 37.54 s.

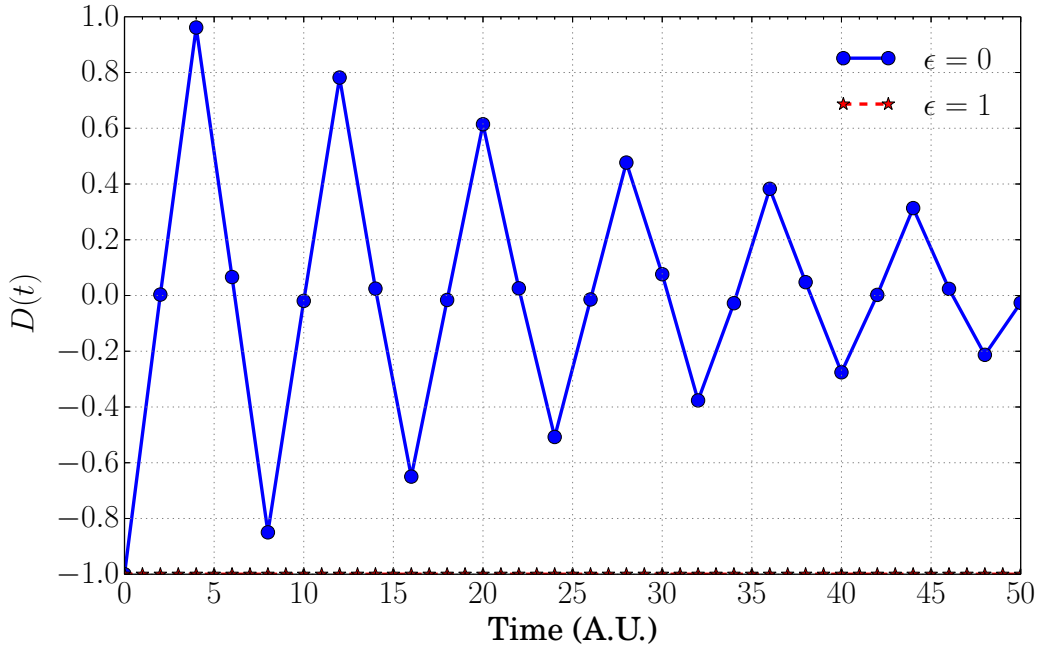


(b) $i = 2$. For the symmetric problem, $h = 0.1$ and MC reps = 5000. For the asymmetric problem, $h = 0.01$ and MC reps = 30000. Computational time symmetric: 1 hrs. 16 min. 0.12 s.; asymmetric: 3 hrs. 44 min. 9.44 s.

Figure 3.19: Spin-boson calculations for symmetric ($\epsilon = 0$) and asymmetric ($\epsilon = 1$) systems. $\alpha = 0.09,\ \beta = 0.25,\ \Delta = (2.5)^{-1}$.



(a) $i = 1$. For the symmetric problem, $h = 0.1$ and MC reps = 5000. For the asymmetric problem, $h = 0.05$ and MC reps = 30000. Computational time symmetric: 1 hrs. 43 min. 32.96 s.; asymmetric: 16 hrs. 58 min. 5.03 s.



(b) $i = 2$. For the symmetric problem, $h = 0.1$ and MC reps = 5000. For the asymmetric problem, $h = 0.05$ and MC reps = 30000. Computational time symmetric: 1 hrs. 52 min. 0.86 s.; asymmetric: 46 min. 17.54 s.

Figure 3.20: Spin-boson calculations for symmetric ($\epsilon = 0$) and asymmetric ($\epsilon = 1$) systems. $\alpha = 0.1,\ \beta = 12.5,\ \Delta = (2.5)^{-1}$.

3.5 Parallelisation and Scaling

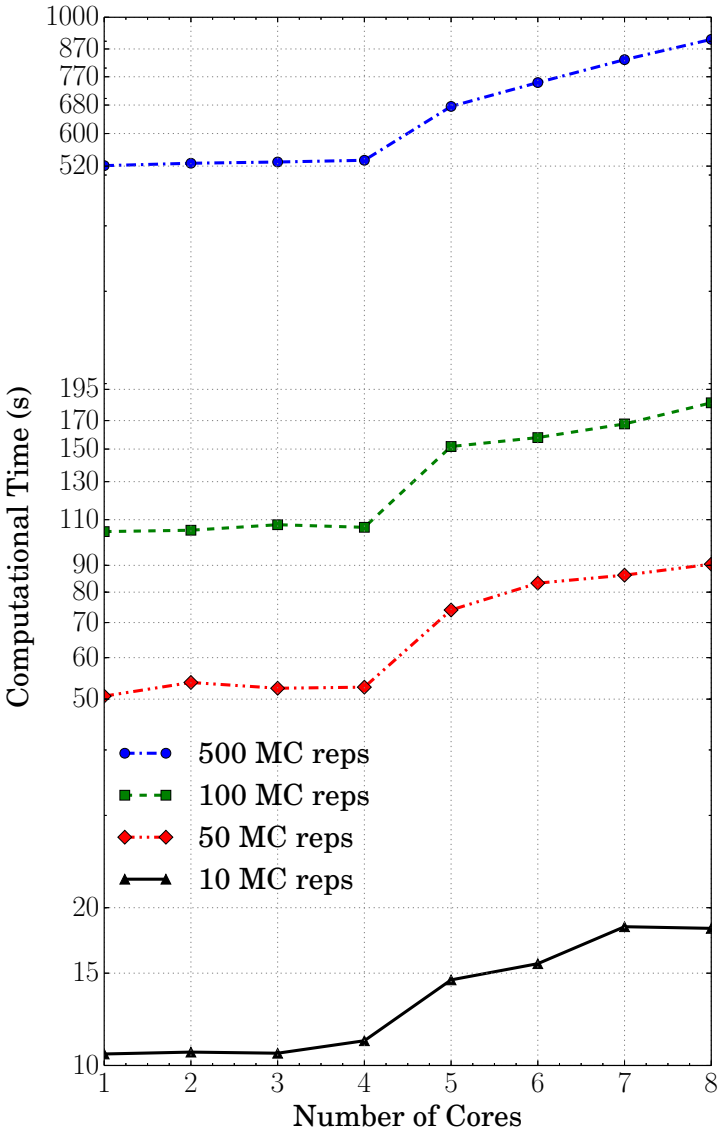


Figure 3.21: Time scaling as a function of the number of cores and MC reps. After 4 cores, Open MPI starts utilising the processor's hyper-threading capabilities. `mpirun` runs copies of the program on the number of specified cores, meaning that the slower copies progressively overwrite the faster ones' files, thus making the slowest execution time the only surviving measurement. This means that $t \propto n^{-1}$, where $t \equiv$ time and $n \equiv \#$ of cores used. It also shows that t scales linearly with the number of MC reps. Test system: single avoided crossing.

Chapter 4

Conclusions

4.1 Conclusion

The most recent version of the MM model was successfully implemented in FORTRAN 2008. It was validated with the three avoided crossing problems defined by Tully [2], and subsequently analysed by Cotton and Miller [1]. Aside from some minor discrepancies attributed to integration step size, RNG, compiler, and number of trajectories used, our results show very good agreement with those found in [1].

Furthermore, new results were obtained for all three avoided crossing problems, where $i = 2$, as well as new parameter sets for the spin-boson model for $i = 1, 2$.

In the case of the three avoided crossing problems, these new results should be verified using a true quantum mechanical model. This is uncharted territory, given that the systems' initial states are excited, rather than ground states. However, it is not unreasonable to assume that these new results are accurate, or at least, as accurate as the results for the ground state. The reason why is because the MM model is a continuous model, so even when $i = 1$, the particle can smoothly move out of the ground state and into the excited one; and its trajectory will smoothly and continuously evolve just as before. So there is no reason why having $i = 2$, should be any different than back-tracing the trajectory of a particle with $f = 2$.

As mentioned in chapter 3, the results of the spin-boson model problems are surprising. We were fortunate to see such a wide variety of behaviours with our choice of parameters. Not only did we see coherent and incoherent oscillations, but we saw the radically different behaviours between the symmetric and asymmetric versions of all problems. The reason for this difference is simple—the energy difference between both states widens, and therefore we get very anisotropic[1] transition probabilities.

[1] When a forward process is not the same as its reverse. For example, flow valves, they allow flow in one direction but not the reverse. Mathematically this can be written as $A \rightarrow B \neq A \leftarrow B$. In the present case this is taken to mean that if x is the probability of transitioning from $A \rightarrow B$, then $A \leftarrow B$ is drastically different from x and not necessarily in the neighbourhood of $1 - x$.

The other surprising (and new) results are the large stability regions observed in one parameter set. Such results, make this parameter set a prime candidate for the study of quantum tunnelling in large systems. Unfortunately, this is non-trivial, because it requires a model which takes quantum tunnelling into consideration.

Lastly, the code's performance and scaling were found to be excellent. Its performance under parallelisation was found to be much better than expected; the results are not affected whatsoever, and the computational time is inversely proportional to the number of cores—this is exactly what we want to see. All that remains in this regard is to parallelise the code itself, rather than manually have to run different copies in different cores and averaging the results.

4.2 Future Work

In order to identify the source of the discrepancies, efforts should be made to lightly modify the code by using different RNGs and seed generators, as well as doing further testing with smaller integration intervals and more MC reps. The code should also be generalised so it can read input files and be applied to arbitrary systems. The use of an adaptive integration algorithm could also be worth investigating. Lastly, attempts must be made to eliminate the need for analytic PES so the model may be applied to real systems. If the code can be sufficiently generalised, it will be added to CFOUR [36], and thus be freely placed at the disposition of the scientific community.

Appendix A

Codes

A.1 Main Codes

The public version of this thesis does not contain the main codes I wrote, because updated and generalised versions of them may become part of future CFOUR [36] releases.

A.1.1 Calling and Printing PES and Their Derivatives

```
program main
  implicit none
  integer, parameter :: es = 2 ! Electronic states.
  call print_pes(es)
end program main

subroutine PES(flag, a, b, c, d, eo, r, h, e, omega, es)
!=========================================================!
! Potential energy surfaces as defined by Miller          !
!---------------------------------------------------------!
! Daniel Celis Garza 16 Jan 2015, 20 Jan 2015             !
!---------------------------------------------------------!
! flag = 1  ->  single avoided crossing (diabatic)        !
! flag = 2  ->  dual avoided crossing (diabatic)          !
! flag = 3  ->  extended coupling (adiabatic)             !
!---------------------------------------------------------!
! h(n)      -> diabatic matrix elements                   !
! e(n/2)    -> adiabatic matrix elements (eigenvalues of  !
!              the diabatic matrix)                       !
!---------------------------------------------------------!
! h(1) = H11    h(2) = H12    h(3) = H21    h(4) = H22    !
! e(1) = E1     e(2) = E2                                 !
!=========================================================!
implicit none
integer es
integer flag
double precision a, b, c, d, eo, r, h(es*es), e(es), omega
! Single Avoided Crossing
if (flag == 1) then
   if (r > 0) then
      h(1) = a*(1-exp(-b*r))
      e(1) = -exp(-d*r**2)*sqrt(c**2+exp(2*d*r**2)*(a*(1-exp(-b*r)))**2)
      e(2) = -e(1)
   else
      h(1) = -a*(1-exp(b*r))
      e(1) = -exp(-d*r**2)*sqrt(c**2+exp(2*d*r**2)*(-a*(1-exp(b*r)))**2)
      e(2) = -e(1)
```

```
   end if
   h(2) = c*exp(-d*r**2)
   h(3) = h(2)
   h(4) = -h(1)
! Double Avoided Crossing
else if (flag == 2) then
   h(1) = 0
   h(2) = c*exp(-d*r**2)
   h(3) = h(2)
   h(4) = -a*exp(-b*r**2) + eo
   e(1) = exp(-(b+d)*r**2)*(-a*exp(d*r**2)+exp((b+d)*r**2)*eo-sqrt(4*c**2*exp(2*b*r**2)&
        +exp(2*d*r**2)*(a-exp(b*r**2)*eo)**2))/2
   e(2) = exp(-(b+d)*r**2)*(-a*exp(d*r**2)+exp((b+d)*r**2)*eo+sqrt(4*c**2*exp(2*b*r**2)&
        +exp(2*d*r**2)*(a-exp(b*r**2)*eo)**2))/2
! Extended Coupling
else if (flag == 3) then
   h(1) = -a
   if (r > 0) then
      h(2) = b*(2-exp(-c*r))
   else
      h(2) = b*exp(c*r)
   end if
   h(3) = h(2)
   h(4) = -h(1)
   omega = atan(2*h(2)/(h(1)-h(4)))/2
   if (r > 0) then
      e(1) = -sqrt(a**2 + (b*(2-exp(-c*r)))**2)
      e(2) = -e(1)
   else
      e(1) = -sqrt(a**2 + (b*exp(c*r))**2)
      e(2) = -e(1)
   end if
end if
end subroutine PES

subroutine dPES(flag, a, b, c, d, r, dh, de, es)
!=============================================================!
! Analytical derivatives of the PES used in Hamilton's eqs    !
!-------------------------------------------------------------!
! Daniel Celis Garza 21 Jan 2015                              !
!=============================================================!
implicit none
integer es
integer flag
double precision a, b, c, d, r, dh(es*es), de(es)
if (flag == 1) then
   if (r > 0) then
      dh(1) = a*b*exp(-b*r)
   else
      dh(1) = a*b*exp(b*r)
   end if
   dh(2) = -2*c*d*r*exp(-d*r**2)
   dh(3) = dh(2)
   dh(4) = -dh(1)
else if (flag == 2) then
   dh(1) = 0.0
   dh(2) = -2*c*d*r*exp(-d*r**2)
   dh(3) = dh(2)
   dh(4) = 2*a*b*r*exp(-b*r**2)
else if (flag == 3) then
   dh(1) = 0
   dh(4) = 0
   if (r > 0) then
      de(1) = -b**2*c*exp(-2*c*r)*(2*exp(c*r)-1)/sqrt(a**2+b**2*(2-exp(-c*r))**2)
      dh(2) = b*c*exp(-c*r)
   else
      de(1) = -b**2*c*exp(2*c*r)/sqrt(a**2+b**2*exp(2*c*r))
      dh(2) = b*c*exp(c*r)
   end if
   de(2) = -de(1)
   dh(3) = dh(2)
end if
end subroutine dPES

subroutine print_PES(es)
! Testing testing all PES
implicit none
integer es
integer flag
double precision a, b, c, d, eo, r, h(es*es), e(es), omega, dh(es*es), de(es)
```

```
do flag = 1, 3
   if (flag == 1) then
      r = -4.0
      a = 0.01
      b = 1.6
      c = 0.005
      d = 1
      open(unit=1, file='sac.dat')
      open(unit=2, file='dsac.dat')
      call PES(flag, a, b, c, d, eo, r, h, e, omega, es)
      call dPES(flag, a, b, c, d, r, dh, de, es)
      write(1,*) r, h(1), h(2), h(3), h(4), e(1), e(2)
      write(2,*) r, dh(1), dh(2), dh(3), dh(4)
      do while(r<4)
         r = r + 0.01
         call PES(flag, a, b, c, d, eo, r, h, e, omega, es)
         call dPES(flag, a, b, c, d, r, dh, de, es)
         write(1,*) r, h(1), h(2), h(3), h(4), e(1), e(2)
         write(2,*) r, dh(1), dh(2), dh(3), dh(4)
      end do
   else if (flag == 2) then
      r  = -8.0
      a  = 0.1
      b  = 0.28
      c  = 0.015
      d  = 0.06
      eo = 0.05
      open(unit=1, file='dac.dat')
      open(unit=2, file='ddac.dat')
      call PES(flag, a, b, c, d, eo, r, h, e, omega, es)
      call dPES(flag, a, b, c, d, r, dh, de, es)
      write(1,*) r, h(1), h(2), h(3), h(4), e(1), e(2)
      write(2,*) r, dh(1), dh(2), dh(3), dh(4)
      do while(r<8)
         r = r + 0.01
         call PES(flag, a, b, c, d, eo, r, h, e, omega, es)
         call dPES(flag, a, b, c, d, r, dh, de, es)
         write(1,*) r, h(1), h(2), h(3), h(4), e(1), e(2)
         write(2,*) r, dh(1), dh(2), dh(3), dh(4)
      end do
   else if (flag == 3) then
      r = -10.0
      a = 6e-4
      b = 0.1
      c = 0.9
      open(unit=1, file='ec.dat')
      open(unit=2, file='dec.dat')
      call PES(flag, a, b, c, d, eo, r, h, e, omega, es)
      call dPES(flag, a, b, c, d, r, dh, de, es)
      write(1,*) r, h(1)*50, h(2), h(3), h(4)*50, e(1), e(2)
      write(2,*) r, dh(1), dh(2), dh(3), dh(4), de(1), de(2)
      do while(r<10)
         r = r + 0.01
         call PES(flag, a, b, c, d, eo, r, h, e, omega, es)
         call dPES(flag, a, b, c, d, r, dh, de, es)
         write(1,*) r, h(1)*50, h(2), h(3), h(4)*50, e(1), e(2)
         write(2,*) r, dh(1), dh(2), dh(3), dh(4), de(1), de(2)
      end do
   end if
end do
end subroutine print_PES
```

A.1.2 Auxiliary Functions Module (Heaviside, Kronecker's Delta, RK4G, Random Seed Initialiser)

```
module functions
implicit none
contains
  function heaviside(x)
    !=======================================================!
    ! Heaviside function                                    !
    ! Daniel Celis Garza 15 Jan 2015                        !
```

```
8     !=====================================================!
9     double precision heaviside, x
10    heaviside = 0.0
11    if (x .ge. 0.0) heaviside = 1.0
12  end function heaviside
13
14  function kdelta(i,j)
15    !=====================================================!
16    ! Kroencker delta function                            !
17    ! Daniel Celis Garza 15 Jan 2015                      !
18    ! delta_ij                                            !
19    !=====================================================!
20    integer i, j
21    double precision kdelta
22    kdelta = 0.0
23    if (i .eq. j) kdelta = 1.0
24  end function kdelta
25
26  subroutine rk4gn(f,ti,tf,xi,xf,n)
27    !=====================================================!
28    ! Solve n first-order ODEs or n/2 second-order.       !
29    ! Runge-Kutta 4 Gill's method                         !
30    ! Daniel Celis Garza 10 Dec. 2014                     !
31    !-----------------------------------------------------!
32    ! f(t,x,dx,n) = parametrised equations of motion      !
33    ! ti          = independent variable @ step i         !
34    ! tf          = independent variable @ step i+1       !
35    ! xi()        = array of dependent variables @ step i !
36    ! xf()        = array of dependent variables @ step i+1 !
37    ! n           = # of parametrised equations of motion !
38    !-----------------------------------------------------!
39    ! j           = counter variable                      !
40    ! h           = time step                             !
41    ! t           = current time                          !
42    ! ki()        = array of Runge-Kutta k's              !
43    !=====================================================!
44    integer n, j
45    double precision ti, tf, h, t, xi(n), xf(n), k1(n), k2(n), k3(n), k4(n), x(n), dx(n), sqr2
46
47    h = tf - ti ! This will be replaced by the error-correcting routine.
48    t = ti
49    sqr2 = sqrt(2.0)
50    ! Calculate k1
51    call f(t,xi,dx,n)             ! Call the equations of motion
52    do concurrent (j = 1: n)      ! Go through all equations of motion
53       k1(j) = h*dx(j)            ! Calculate the value of k1 for each equation of motion
54       x(j)  = xi(j) + k1(j)/2.0 ! Calculate the next value of x for each equation
55    end do                        ! (to be used in the next ki)
56
57    ! Calculate k2
58    call f(t+h/2.0,x,dx,n)
59    do concurrent (j = 1: n)
60       k2(j) = h*dx(j)
61       x(j)  = xi(j) + (sqr2-1.0)*k1(j)/2.0 + (1.0-sqr2/2.0)*k2(j)
62    end do
63
64    ! Calculate k3
65    call f(t+h/2.0,x,dx,n)
66    do concurrent (j = 1: n)
67       k3(j) = h*dx(j)
68       x(j)  = xi(j) - sqr2*k2(j)/2.0 + (1.0+sqr2/2.0)*k3(j)
69    end do
70
71    ! Calculate k4 and xf
72    call f(t+h,x,dx,n)
73    do concurrent (j = 1: n)
74       k4(j) = h*dx(j)
75       xf(j)  = xi(j) + (k1(j) + (2.0-sqr2)*k2(j) + (2+sqr2)*k3(j) + k4(j))/6.0
76    end do
77  end subroutine rk4gn
78
79  subroutine rk4gnsb(f,ti,tf,xi,xf,w,c,n,es)
80    !=====================================================!
81    ! Solve n first-order ODEs or n/2 second-order.       !
82    ! Runge-Kutta 4 Gill's method                         !
83    ! Daniel Celis Garza 10 Dec. 2014                     !
84    !-----------------------------------------------------!
85    ! f(t,x,dx,n) = parametrised equations of motion      !
86    ! ti          = independent variable @ step i         !
87    ! tf          = independent variable @ step i+1       !
```

```
88      ! xi()        = array of dependent variables @ step i   !
89      ! xf()        = array of dependent variables @ step i+1 !
90      ! n           = # of parametrised equations of motion   !
91      !-------------------------------------------------------!
92      ! j           = counter variable                        !
93      ! h           = time step                               !
94      ! t           = current time                            !
95      ! ki()        = array of Runge-Kutta k's                !
96      !=======================================================!
97      implicit none
98      integer n, es, j
99      double precision ti,tf,h,t,xi(n),xf(n),k1(n),k2(n),k3(n),k4(n),x(n),dx(n),sqr2,w(n),c(n)
100
101     h = tf - ti ! This will be replaced by the error-correcting routine.
102     t = ti
103     sqr2 = sqrt(2.0)
104     ! Calculate k1
105     call f(t,xi,dx,w,c,n,es)      ! Call the equations of motion
106     do concurrent (j = 1: n)      ! Go through all equations of motion
107        k1(j) = h*dx(j)            ! Calculate the value of k1 for each equation of motion
108        x(j)  = xi(j) + k1(j)/2.0 ! Calculate the next value of x for each equation
109     end do                        ! (to be used in the next ki)
110
111     ! Calculate k2
112     call f(t+h/2.0,x,dx,w,c,n,es)
113     do concurrent (j = 1: n)
114        k2(j) = h*dx(j)
115        x(j)  = xi(j) + (sqr2-1.0)*k1(j)/2.0 + (1.0-sqr2/2.0)*k2(j)
116     end do
117
118     ! Calculate k3
119     call f(t+h/2.0,x,dx,w,c,n,es)
120     do concurrent (j = 1: n)
121        k3(j) = h*dx(j)
122        x(j)  = xi(j) - sqr2*k2(j)/2.0 + (1.0+sqr2/2.0)*k3(j)
123     end do
124
125     ! Calculate k4 and xf
126     call f(t+h,x,dx,w,c,n,es)
127     do concurrent (j = 1: n)
128        k4(j) = h*dx(j)
129        xf(j)  = xi(j) + (k1(j) + (2.0-sqr2)*k2(j) + (2+sqr2)*k3(j) + k4(j))/6.0
130     end do
131   end subroutine rk4gnsb
132
133   subroutine init_random_seed()
134     !=======================================================!
135     !-----------Standard F95 Random seed routine------------!
136     ! rndnum can be any n x m array or a scalar             !
137     ! REAL :: rndnum(n,m)                                   !
138     ! CALL init_random_seed()                               !
139     ! CALL RANDOM_NUMBER(rndnum)                            !
140     !=======================================================!
141     use iso_fortran_env, only: int64
142     implicit none
143     integer, allocatable :: seed(:)
144     integer :: i, n, un, istat, dt(8), pid
145     integer(int64) :: t
146
147     call random_seed(size = n)
148     allocate(seed(n))
149     ! First try if the OS provides a random number generator
150     open(newunit=un, file="/dev/urandom", access="stream", &
151          form="unformatted", action="read", status="old", iostat=istat)
152     if (istat == 0) then
153        read(un) seed
154        close(un)
155     else
156        ! Fallback to XOR:ing the current time and pid. The PID is
157        ! useful in case one launches multiple instances of the same
158        ! program in parallel.
159        call system_clock(t)
160        if (t == 0) then
161           call date_and_time(values=dt)
162           t = (dt(1) - 1970) * 365_int64 * 24 * 60 * 60 * 1000 &
163                + dt(2) * 31_int64 * 24 * 60 * 60 * 1000 &
164                + dt(3) * 24_int64 * 60 * 60 * 1000 &
165                + dt(5) * 60 * 60 * 1000 &
166                + dt(6) * 60 * 1000 + dt(7) * 1000 &
```

```
                + dt(8)
        end if
        pid = getpid()
        t = ieor(t, int(pid, kind(t)))
        do i = 1, n
           seed(i) = lcg(t)
        end do
    end if
    call random_seed(put=seed)
  contains
    ! This simple PRNG might not be good enough for real work, but is
    ! sufficient for seeding a better PRNG.
    function lcg(s)
      integer :: lcg
      integer(int64) :: s
      if (s == 0) then
         s = 104729
      else
         s = mod(s, 4294967296_int64)
      end if
      s = mod(s * 279470273_int64, 4294967291_int64)
      lcg = int(mod(s, int(huge(0), int64)), kind(0))
    end function lcg
  end subroutine init_random_seed
end module functions
```

A.2 Python 3.4 Scripts

A.2.1 Plot PES

```
"""
PES Plotting Script.
Author: Daniel Celis Garza
"""

import numpy as np
import matplotlib
matplotlib.use('Agg') # Lets us save in .svg format.
import matplotlib.pyplot as plt
from matplotlib import ticker

plt.close('all') # Close all figures when replotting.
# Plotting parameters for use at scale = 0.5 with the thesis template.
params = {'backend': 'ps',
          'axes.labelsize': 24,
          'axes.titlesize': 24,
          'text.fontsize': 24,
          'legend.fontsize': 24,
          'xtick.labelsize': 24,
          'ytick.labelsize': 24,
          'text.usetex': True, # LaTeX Masterrace Huuuh haaaah!
          'font.family': 'serif',
          'mathtext.fontset': 'custom'}
plt.rcParams.update(params)
# LaTeX font activated by " r'' ". Accepts all environments.

dpt = '../data/pes/'
ipt = '../images/'
ext = 'svg'

# Plot diabatic PES.
def plotd(data,legend,b,xlim,name,ext):
    x, y0, y1, y2 = data[:,0], data[:,1], data[:,2], data[:,4]
    # Creating figure.
    fig, ax = plt.subplots(figsize=(12,7.42)) # Golden ratio ;)
    # Plotting PES.
    ax.plot(x, y0,label=legend[0], ls = '-', lw = 2.5) # Continuous line
    ax.plot(x, y1,label=legend[1], ls = '--', dashes=[8, 4, 2, 4, 2, 4], lw = 2.5) # Custom dashes.
    ax.plot(x, y2,label=legend[2], ls = '--', lw = 2.5) # Dashed line.
    # Adding grid.
```

```
    ax.grid(b=True, which='major', color='0.5',linestyle=':') # Dotted line.
    # Axis labels.
    ax.set_xlabel(r'Position ($\mathbf{R}$)')
    ax.set_ylabel(r'Energy (A.U.)')
    # Scientific notation on y-axis for the win.
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    # Minor and major tick locators.
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0])) # Major x-tick every 1.
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=4)) # 4 Minor x-tick per major tick.
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1])) # Major y-tick every 0.002.
    # x-range
    ax.set_xlim(xlim)
    # Legend options (smart legend positioning, deciding whether to remove frame or not).
    ax.legend(loc=0,frameon=False,fancybox=True,framealpha=1)
    # Adjusting the plot so the bottom legend fits into the figure.
    plt.subplots_adjust(bottom=0.12)
    # Saving into a .svg so it can be converted into anything else with Inkscape.
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

# Plot adiabatic PES.
def plota(data,legend,b,xlim,name,ext):
    x, y0, y1 = data[:,0], data[:,5], data[:,6]
    # Creating figure.
    fig, ax = plt.subplots(figsize=(12,7.42)) # Golden ratio ;)
    # Plotting PES.
    ax.plot(x, y0,label=legend[0], ls = '-', c = 'b', lw = 2.5) # Continuous line
    # Custom dashes.
    ax.plot(x, y1,label=legend[1], ls = '--', c = 'r', dashes=[8, 4, 2, 4, 2, 4], lw = 2.5)
    # Adding grid.
    ax.grid(b=True, which='major', color='0.5',linestyle=':') # Dotted line.
    # Axis labels.
    ax.set_xlabel(r'Position ($\mathbf{R}$)')
    ax.set_ylabel(r'Energy (A.U.)')
    # Scientific notation on y-axis for the win.
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    # Minor and major tick locators.
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0])) # Major x-tick every 1.
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=4)) # 4 Minor x-tick per major tick.
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1])) # Major y-tick every 0.002.
    # x-range
    ax.set_xlim(xlim)
    # Legend options (smart legend positioning, deciding whether to remove frame or not).
    ax.legend(loc=0,frameon=False,fancybox=True,framealpha=1)
    # Adjusting the plot so the bottom legend fits into the figure.
    plt.subplots_adjust(bottom=0.12)
    # Saving into a .svg so it can be converted into anything else with Inkscape.
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

# Plot E2 - E1
def plotad(data,legend,b,xlim,name,ext):
    x, y0, y1 = data[:,0], data[:,5], data[:,6]
    # Creating figure.
    fig, ax = plt.subplots(figsize=(12,7.42)) # Golden ratio ;)
    # Plotting PES.
    ax.plot(x, y1-y0,label=legend[0], ls = '-', c = 'b', lw = 2.5) # Continuous line
    # Adding grid.
    ax.grid(b=True, which='major', color='0.5',linestyle=':') # Dotted line.
    # Axis labels.
    ax.set_xlabel(r'Position ($\mathbf{R}$)')
    ax.set_ylabel(r'$\Delta$ Energy (A.U.)')
    # Scientific notation on y-axis for the win.
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    # Minor and major tick locators.
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0])) # Major x-tick every 1.
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=4)) # 4 Minor x-tick per major tick.
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1])) # Major y-tick every 0.002.
    # x-range
    ax.set_xlim(xlim)
    # Legend options (smart legend positioning, deciding whether to remove frame or not).
    ax.legend(loc=0,frameon=False,fancybox=True,framealpha=1)
    # Adjusting the plot so the bottom legend fits into the figure.
```

```
    plt.subplots_adjust(bottom=0.12)
    # Saving into a .svg so it can be converted into anything else with Inkscape.
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

# Single Avoided Crossing Diabatic
data   = np.loadtxt(dpt+'sac.dat')
legend = [r'$ H_{11} $',r'$ H_{12} = H_{21} $',r'$ H_{22} $']
b      = [1,0.002]
xlim   = [-4,4]
name   = ipt+'scpes'
plotd(data,legend,b,xlim,name,ext)

# Single Avoided Crossing Adiabatic
legend = [r'$ E_{1} $',r'$ E_{2}']
b      = [1,0.002]
xlim   = [-4,4]
name   = ipt+'ascpes'
plota(data,legend,b,xlim,name,ext)

# Single Avoided Crossing E2 - E1
legend = [r'$ E_{2} - E_{1} $']
b      = [1,0.002]
xlim   = [-4,4]
name   = ipt+'del_ascpes'
plotad(data,legend,b,xlim,name,ext)

# Single Avoided Crossing Diabatic Derivatives
data = np.loadtxt(dpt+'dsac.dat')
legend = [r'$ \frac{\partial H_{11}}{\partial R} $',
          r'$ \frac{\partial H_{12}}{\partial R} = \frac{\partial H_{21}}{\partial R} $',
          r'$ \frac{\partial H_{22}}{\partial R} $']
b      = [1,0.005]
xlim   = [-4,4]
name   = ipt+'dscpes'
plotd(data,legend,b,xlim,name,ext)

# ---------------------------------------- #

# Double Avoided Crossing Diabatic
data = np.loadtxt(dpt+'dac.dat')
legend = [r'$ H_{11} $',r'$ H_{12} = H_{21} $',r'$ H_{22} $']
b      = [2,0.01]
xlim   = [-8,8]
name   = ipt+'dcpes'
plotd(data,legend,b,xlim,name,ext)

# Double Avoided Crossing Adiabatic
legend = [r'$ E_{1} $',r'$ E_{2}']
b      = [2,0.01]
xlim   = [-8,8]
name   = ipt+'adcpes'
plota(data,legend,b,xlim,name,ext)

# Double Avoided Crossing E2 - E1
legend = [r'$ E_{2} - E_{1} $']
b      = [1,0.005]
xlim   = [-8,8]
name   = ipt+'del_adcpes'
plotad(data,legend,b,xlim,name,ext)

# Double Avoided Crossing Diabatic Derivatives
data = np.loadtxt(dpt+'ddac.dat')
legend = [r'$ \frac{\partial H_{11}}{\partial R} $',
          r'$ \frac{\partial H_{12}}{\partial R} = \frac{\partial H_{21}}{\partial R} $',
          r'$ \frac{\partial H_{22}}{\partial R} $']
b      = [2,0.01]
xlim   = [-8,8]
name   = ipt+'ddcpes'
plotd(data,legend,b,xlim,name,ext)

# ---------------------------------------- #

# Extended Coupling Diabatic
data = np.loadtxt(dpt+'ec.dat')
legend = [r'$ H_{11} \times 50 $',r'$ H_{12} = H_{21} $',r'$ H_{22} \times 50 $']
b      = [2,0.025]
xlim   = [-10,10]
name   = ipt+'ecpes'
plotd(data,legend,b,xlim,name,ext)

```

```
# Extended Coupling Adiabatic
legend = [r'$ E_{1} $',r'$ E_{2}']
b      = [2,0.05]
xlim   = [-10,10]
name   = ipt+'aecpes'
plota(data,legend,b,xlim,name,ext)

# Double Avoided Crossing E2 - E1
legend = [r'$ E_{2} - E_{1} $']
b      = [2,0.05]
xlim   = [-10,10]
name   = ipt+'del_aecpes'
plotad(data,legend,b,xlim,name,ext)

# Extended Coupling Diabatic Derivatives
data = np.loadtxt(dpt+'dec.dat')
legend = [r'$ \frac{\partial H_{11}}{\partial R} = \frac{\partial H_{22}}{\partial R} $',
          r'$ \frac{\partial H_{12}}{\partial R} = \frac{\partial H_{21}}{\partial R} $',
          r'$ \frac{\partial H_{22}}{\partial R} = \frac{\partial H_{11}}{\partial R} $']
b      = [2,0.01]
xlim   = [-10,10]
name   = ipt+'decpes'
plotd(data,legend,b,xlim,name,ext)

# Extended Coupling Adiabatic Derivatives
legend = [r'$ \frac{\partial E_{1}}{\partial R} $',r'$ \frac{\partial E_{2}}{\partial R} $']
b      = [2,0.02]
xlim   = [-10,10]
name   = ipt+'daecpes'
plota(data,legend,b,xlim,name,ext)
```

A.2.2 Plot Transition Probabilities

```
"""
Probabilities Plotting Script.
Author: Daniel Celis Garza
"""

import numpy as np
import matplotlib
matplotlib.use('Agg') # Use .svg format.
import matplotlib.pyplot as plt
from matplotlib import ticker

plt.close('all')
# Plotting parameters
params = {'backend': 'ps',
          'axes.labelsize': 24,
          'axes.titlesize': 24,
          'text.fontsize': 24,
          'legend.fontsize': 24,
          'xtick.labelsize': 24,
          'ytick.labelsize': 24,
          'text.usetex': True,
          'font.family': 'serif',
          'mathtext.fontset': 'custom'}
plt.rcParams.update(params)

scp = '../data/single_avoided_crossing/'
dcp = '../data/double_avoided_crossing/'
ecp = '../data/extended_coupling/'
sbp = '../data/spin_boson/'
ipt = '../images/'
ext = 'eps'

def plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext):
    x, y0, y1, y2, y3 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(x, y0,label=r'$ R_{1\leftarrow 1} $', ls = '-', c = 'b', lw = 2.5,
            marker = 'o', markersize = 10)
    ax.plot(x, y1,label=r'$ R_{2\leftarrow 1} $', ls = '-.', c = 'g', lw = 2.5,
            marker = 's', markersize = 10)
    ax.plot(x, y2,label=r'$ T_{1\leftarrow 1} $', ls = '--', c = 'r', lw = 2.5,
            marker = 'd', markersize = 10)
```

```
    ax.plot(x, y3,label=r'$ T_{2\leftarrow 1} $', ls = '--', c = 'k', lw = 2.5,
            dashes=[8, 4, 2, 4, 2, 4], marker = '*', markersize = 10)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    plt.legend(bbox_to_anchor=lpos,
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=0.1))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

def plotsb(data1,data2,b,xlim,ylim,name,ext):
    x0, y0, x1, y1 = data1[:,0], data1[:,1], data2[:,0], data2[:,1]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(x0, y0,label=r'$ \epsilon = 0 $', ls = '-', c = 'b', lw = 2.5,
           marker = 'o', markersize = 10)
    ax.plot(x1, y1,label=r'$ \epsilon = 1 $', ls = '--', c = 'r', lw = 2.5,
           marker = '*', markersize = 10)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(r'Time (A.U.)')
    ax.set_ylabel(r'$ D(t) $')
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    plt.legend(loc=0,#bbox_to_anchor=lpos
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0]))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1]))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

def plott(data,b,xlim,ylim,name,ext):
    t, r, p, n1, n2 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(t, r,label=r'$ R $', ls = '-.', c = 'b', lw = 2.5)
    ax.plot(t, p,label=r'$ P $', ls = '--', dashes=[8, 4, 2, 4, 2, 4], c = 'g', lw = 2.5)
    ax.plot(t, n1,label=r'$ n_{1} + \gamma $', ls = '--', c = 'r', lw = 2.5)
    ax.plot(t, n2,label=r'$ n_{2} + \gamma$', ls = '-', c = 'k', lw = 2.5)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(r'Time (A.U.)')
    ax.set_ylabel(r'Value (A.U.)')
#    formatter = ticker.ScalarFormatter(useMathText=True)
#    formatter.set_scientific(True)
#    formatter.set_powerlimits((-1,1))
#    ax.yaxis.set_major_formatter(formatter)
    plt.legend(loc=0,#bbox_to_anchor=lpos
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0]))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1]))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
#    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

def plott2(data,b,lpos,xlim,ylim,name,ext):
    t, r, p, n1, n2 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(t, r,label=r'$ R $', ls = '-.', c = 'b', lw = 2.5)
    ax.plot(t, p,label=r'$ P $', ls = '--', dashes=[8, 4, 2, 4, 2, 4], c = 'g', lw = 2.5)
    ax.plot(t, n1,label=r'$ n_{1} + \gamma$', ls = '--', c = 'r', lw = 2.5)
    ax.plot(t, n2,label=r'$ n_{2} + \gamma$', ls = '-', c = 'k', lw = 2.5)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(r'Time (A.U.)')
    ax.set_ylabel(r'Value (A.U.)')
#    formatter = ticker.ScalarFormatter(useMathText=True)
#    formatter.set_scientific(True)
#    formatter.set_powerlimits((-1,1))
```

```
#    ax.yaxis.set_major_formatter(formatter)
    plt.legend(bbox_to_anchor=lpos,
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0]))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1]))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
#    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

def plote(data,b,xlim,ylim,name,ext):
    t, n1, n2 = data[:,0], data[:,3], data[:,4]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(t, n1,label=r'$ n_{1} + \gamma$', ls = '--', c = 'r', lw = 2.5)
    ax.plot(t, n2,label=r'$ n_{2} + \gamma$', ls = '-', c = 'k', lw = 2.5)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(r'Time (A.U.)')
    ax.set_ylabel(r'Value (A.U.)')
#    formatter = ticker.ScalarFormatter(useMathText=True)
#    formatter.set_scientific(True)
#    formatter.set_powerlimits((-1,1))
#    ax.yaxis.set_major_formatter(formatter)
    plt.legend(loc=0,#bbox_to_anchor=lpos
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b[0]))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=b[1]))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
#    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

# Trajectories
data = np.loadtxt(scp+'sc_traj_R11.dat')
name = ipt+'sc_traj_r11'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [min(data[:,1]),max(data[:,3])+0.1]
b = [1000,0.5]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'sc_traj_r11_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0.3,1.7]
b = [1000,0.1]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(scp+'sc_traj_T12.dat')
name = ipt+'sc_traj_t12'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [min(data[:,1]),max(data[:,2])+0.1]
b = [500,1]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'sc_traj_t12_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.15]
b = [500,0.1]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(scp+'sc_traj_T11.dat')
name = ipt+'sc_traj_t11'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [min(data[:,1]),max(data[:,2])+1]
b = [100,5]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'sc_traj_t11_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0.4,1.3]
b = [100,0.1]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(scp+'sc_traj_R22.dat')
name = ipt+'sc_traj_r22'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-5,5.5]
b = [5000,1]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'sc_traj_r22_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,2]
b = [5000,0.2]
```

```
202 plote(data,b,xlim,ylim,name,ext)
203
204 data = np.loadtxt(scp+'sc_traj_T21.dat')
205 name = ipt+'sc_traj_t21'
206 xlim = [min(data[:,0]),max(data[:,0])]
207 ylim = [min(data[:,1]),max(data[:,2])+0.1]
208 b = [200,1]
209 plott(data,b,xlim,ylim,name,ext)
210 name = ipt+'sc_traj_t21_e'
211 xlim = [min(data[:,0]),max(data[:,0])]
212 ylim = [0,1.7]
213 b = [200,0.2]
214 plote(data,b,xlim,ylim,name,ext)
215
216 data = np.loadtxt(scp+'sc_traj_T22.dat')
217 name = ipt+'sc_traj_t22'
218 xlim = [min(data[:,0]),max(data[:,0])]
219 ylim = [min(data[:,1]),max(data[:,2])+1]
220 b = [100,3]
221 plott(data,b,xlim,ylim,name,ext)
222 name = ipt+'sc_traj_t22_e'
223 xlim = [min(data[:,0]),max(data[:,0])]
224 ylim = [0.55,1.25]
225 b = [100,0.2]
226 plote(data,b,xlim,ylim,name,ext)
227
228 data = np.loadtxt(dcp+'dc_traj_T11.dat')
229 name = ipt+'dc_traj_t11'
230 xlim = [min(data[:,0]),max(data[:,0])]
231 ylim = [min(data[:,1]),20]
232 b = [200,2]
233 lpos = (0.3,0.67)
234 plott2(data,b,lpos,xlim,ylim,name,ext)
235 name = ipt+'dc_traj_t11_e'
236 xlim = [min(data[:,0]),max(data[:,0])]
237 ylim = [0,1.8]
238 b = [200,0.2]
239 plote(data,b,xlim,ylim,name,ext)
240
241 data = np.loadtxt(dcp+'dc_traj_T12.dat')
242 name = ipt+'dc_traj_t12'
243 xlim = [min(data[:,0]),max(data[:,0])]
244 ylim = [-8,28]
245 b = [200,3]
246 lpos = (0.3,0.67)
247 plott2(data,b,lpos,xlim,ylim,name,ext)
248 name = ipt+'dc_traj_t12_e'
249 xlim = [min(data[:,0]),max(data[:,0])]
250 ylim = [0,2.2]
251 b = [200,0.2]
252 plote(data,b,xlim,ylim,name,ext)
253
254 data = np.loadtxt(dcp+'dc_traj_T21.dat')
255 name = ipt+'dc_traj_t21'
256 xlim = [min(data[:,0]),max(data[:,0])]
257 ylim = [min(data[:,1]),20]
258 b = [200,2]
259 lpos = (0.3,0.7)
260 plott2(data,b,lpos,xlim,ylim,name,ext)
261 name = ipt+'dc_traj_t21_e'
262 xlim = [min(data[:,0]),max(data[:,0])]
263 ylim = [0,2.2]
264 b = [200,0.2]
265 plote(data,b,xlim,ylim,name,ext)
266
267 data = np.loadtxt(dcp+'dc_traj_T22.dat')
268 name = ipt+'dc_traj_t22'
269 xlim = [min(data[:,0]),max(data[:,0])]
270 ylim = [-8,27]
271 b = [200,3]
272 lpos = (0.3,0.67)
273 plott2(data,b,lpos,xlim,ylim,name,ext)
274 name = ipt+'dc_traj_t22_e'
275 xlim = [min(data[:,0]),max(data[:,0])]
276 ylim = [0,1.6]
277 b = [200,0.2]
278 plote(data,b,xlim,ylim,name,ext)
279
280 data = np.loadtxt(dcp+'dc_traj_low_momentum.dat')
281 name = ipt+'dc_traj_low_momentum'
```

```
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [min(data[:,1]),22]
b = [5000,2]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'dc_traj_low_momentum_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.6]
b = [5000,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(dcp+'dc_traj_low_momentum_2.dat')
name = ipt+'dc_traj_low_momentum_2'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-8,2]
b = [2500,1]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'dc_traj_low_momentum_e_2'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0.1,1.65]
b = [2500,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_R11.dat')
name = ipt+'ec_traj_r11'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-16,16]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_r11_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.6]
b = [250,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_R12.dat')
name = ipt+'ec_traj_r12'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-13.5,13.5]
b = [500,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_r12_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.8]
b = [500,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_R21.dat')
name = ipt+'ec_traj_r21'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-16,16]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_r21_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.7]
b = [250,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_R22.dat')
name = ipt+'ec_traj_r22'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-13.5,13.5]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_r22_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.75]
b = [250,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_T11.dat')
name = ipt+'ec_traj_t11'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [min(data[:,1]),27]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_t11_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.7]
b = [250,0.2]
```

```
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_T12.dat')
name = ipt+'ec_traj_t12'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-9,max(data[:,2])+1]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_t12_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.9]
b = [250,0.2]
plote(data,b,xlim,ylim,name,ext)
#
data = np.loadtxt(ecp+'ec_traj_T21.dat')
name = ipt+'ec_traj_t21'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [min(data[:,1]),31]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_t21_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,2]
b = [250,0.2]
plote(data,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_traj_T22.dat')
name = ipt+'ec_traj_t22'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [-9,max(data[:,2])+1]
b = [250,3]
plott(data,b,xlim,ylim,name,ext)
name = ipt+'ec_traj_t22_e'
xlim = [min(data[:,0]),max(data[:,0])]
ylim = [0,1.6]
b = [250,0.2]
plote(data,b,xlim,ylim,name,ext)

# Single Avoided Crossing
data = np.loadtxt(scp+'sc_prob_1o5ip_15000.dat')
xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.52, 0.7)
b = 5
xlim = [0,30]
ylim = [0,1]
name = ipt+'sc_prob_1o5ip'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

data = np.loadtxt(scp+'sc_prob_1o5ip_15000.dat')
data1 = np.loadtxt(scp+'sc_prob_1oip_15000.dat')

x, y0, y1, y2, y3 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
x1, y01, y11, y21, y31 = data1[:,0], data1[:,1], data1[:,2], data1[:,3], data1[:,4]
fig, ax = plt.subplots(figsize=(12,7.42))
ax.plot(x, y0,label=r'$ R_{1\leftarrow 1} $', ls = '-', lw = 2.5, marker = 'o', markersize = 10)
ax.plot(x1, y01,label=r'$ R_{1\leftarrow 1} $', ls = '-', lw = 2.5, marker = 'o', markersize = 10)
ax.plot(x, y1,label=r'$ R_{2\leftarrow 1} $', ls = '-.', lw = 2.5, marker = 's', markersize = 10)
ax.plot(x1, y11,label=r'$ R_{2\leftarrow 1} $', ls = '-.', lw = 2.5, marker = 's', markersize = 10)
ax.plot(x, y2,label=r'$ T_{1\leftarrow 1} $', ls = '--', lw = 2.5, marker = 'd', markersize = 10)
ax.plot(x1, y21,label=r'$ T_{1\leftarrow 1} $', ls = '--', lw = 2.5, marker = 'd', markersize = 10)
ax.plot(x, y3,label=r'$ T_{2\leftarrow 1} $', ls = '--', dashes=[8, 4, 2, 4, 2, 4], lw = 2.5,
        marker = '*', markersize = 10)
ax.plot(x1, y31,label=r'$ T_{2\leftarrow 1} $', ls = '--', dashes=[8, 4, 2, 4, 2, 4], lw = 2.5,
        marker = '*', markersize = 10)
ax.grid(b=True, which='major', color='0.5',linestyle=':')
ax.set_xlabel(r'Initial Nuclear Momentum ($P_{i}$)')
ax.set_ylabel(r'Probability')
formatter = ticker.ScalarFormatter(useMathText=True)
formatter.set_scientific(True)
formatter.set_powerlimits((-1,1))
ax.yaxis.set_major_formatter(formatter)
plt.legend(bbox_to_anchor=(0.49, 0.8),
           bbox_transform=plt.gcf().transFigure, frameon=False)
ax.xaxis.set_major_locator(ticker.MultipleLocator(base=5))
ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
ax.yaxis.set_major_locator(ticker.MultipleLocator(base=0.1))
ax.set_xlim([0,30])
ax.set_ylim([0,1])
plt.subplots_adjust(bottom=0.12)
```

```
plt.savefig(ipt+'sc_prob_1oip_vs_1o5ip.eps', format='eps', dpi=1200)

# Double Avoided Crossing
data = np.loadtxt(dcp+'dc_prob_1o5ip.dat')
xlabel = r'$\ln{(E_{i})}$'
ylabel = r'Probability'
lpos = (0.4, 0.7)
b = 1
xlim = [-4.2,1]
ylim = [0,1]
name = ipt+'dc_prob_1o5ip'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

# Extended Coupling
data = np.loadtxt(ecp+'ec_prob_1o5ip.dat')
xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.35, 0.6)
b = 2
xlim = [2,30]
ylim = [0,1]
name = ipt+'ec_prob_1o5ip'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

# Spin-Boson
data1 = np.loadtxt(sbp+'sb_prob_exp1A1.dat')
data2 = np.loadtxt(sbp+'sb_prob_exp1B1.dat')
b = [5,0.2]
xlim = [0,40]
ylim = [-1,1]
name = ipt+'spin_boson_e11'
plotsb(data1,data2,b,xlim,ylim,name,ext)

data1 = np.loadtxt(sbp+'sb_prob_exp1A2.dat')
data2 = np.loadtxt(sbp+'sb_prob_exp1B2.dat')
b = [5,0.2]
xlim = [0,40]
ylim = [-1,1]
name = ipt+'spin_boson_e12'
plotsb(data1,data2,b,xlim,ylim,name,ext)

data1 = np.loadtxt(sbp+'sb_prob_exp2A1.dat')
data2 = np.loadtxt(sbp+'sb_prob_exp2B1.dat')
b = [5,0.2]
xlim = [0,50]
ylim = [-1,1]
name = ipt+'spin_boson_e21'
plotsb(data1,data2,b,xlim,ylim,name,ext)

data1 = np.loadtxt(sbp+'sb_prob_exp2A2.dat')
data2 = np.loadtxt(sbp+'sb_prob_exp2B2.dat')
b = [5,0.2]
xlim = [0,50]
ylim = [-1,1]
name = ipt+'spin_boson_e22'
plotsb(data1,data2,b,xlim,ylim,name,ext)
```

A.2.3 Plot Transition Probabilities (Large Integration Step)

```
"""
Probabilities Plotting Script.
Author: Daniel Celis Garza
"""

import numpy as np
import matplotlib
matplotlib.use('Agg') # Use .svg format.
import matplotlib.pyplot as plt
from matplotlib import ticker

plt.close('all')
# Plotting parameters
params = {'backend': 'ps',
          'axes.labelsize': 24,
```

```
          'axes.titlesize': 24,
          'text.fontsize': 24,
          'legend.fontsize': 24,
          'xtick.labelsize': 24,
          'ytick.labelsize': 24,
          'text.usetex': True,
          'font.family': 'serif',
          'mathtext.fontset': 'custom'}
plt.rcParams.update(params)

scp = '../data/single_avoided_crossing/'
dcp = '../data/double_avoided_crossing/'
ecp = '../data/extended_coupling/'
ipt = '../images/'
ext = 'eps'

def plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext):
    x, y0, y1, y2, y3 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(x, y0,label=r'$ R_{1\leftarrow 1} $', ls = '-', c = 'b', lw = 2.5,
            marker = 'o', markersize = 10)
    ax.plot(x, y1,label=r'$ R_{2\leftarrow 1} $', ls = '-.', c = 'g', lw = 2.5,
            marker = 's', markersize = 10)
    ax.plot(x, y2,label=r'$ T_{1\leftarrow 1} $', ls = '--', c = 'r', lw = 2.5,
            marker = 'd', markersize = 10)
    ax.plot(x, y3,label=r'$ T_{2\leftarrow 1} $', ls = '--', c = 'k', lw = 2.5,
            dashes=[8, 4, 2, 4, 2, 4], marker = '*', markersize = 10)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    plt.legend(bbox_to_anchor=lpos,
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=0.1))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

def plot2(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext):
    x, y0, y1, y2, y3 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
    fig, ax = plt.subplots(figsize=(12,7.42))
    ax.plot(x, y0,label=r'$ R_{1\leftarrow 2} $', ls = '-', c = 'b', lw = 2.5,
            marker = 'o', markersize = 10)
    ax.plot(x, y1,label=r'$ R_{2\leftarrow 2} $', ls = '-.', c = 'g', lw = 2.5,
            marker = 's', markersize = 10)
    ax.plot(x, y2,label=r'$ T_{1\leftarrow 2} $', ls = '--', c = 'r', lw = 2.5,
            marker = 'd', markersize = 10)
    ax.plot(x, y3,label=r'$ T_{2\leftarrow 2} $', ls = '--', c = 'k', lw = 2.5,
            dashes=[8, 4, 2, 4, 2, 4], marker = '*', markersize = 10)
    ax.grid(b=True, which='major', color='0.5',linestyle=':')
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    formatter = ticker.ScalarFormatter(useMathText=True)
    formatter.set_scientific(True)
    formatter.set_powerlimits((-1,1))
    ax.yaxis.set_major_formatter(formatter)
    plt.legend(bbox_to_anchor=lpos,
           bbox_transform=plt.gcf().transFigure, frameon=False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b))
    ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(base=0.1))
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    plt.subplots_adjust(bottom=0.12)
    plt.savefig(name+'.'+ext, format=ext, dpi=1200)

data = np.loadtxt(scp+'sc_prob_lh.dat')
xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.52, 0.7)
b = 5
xlim = [0,30]
ylim = [0,1]
name = ipt+'sc_prob_lh'
```

```
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

# Initial state = 2
data = np.loadtxt(scp+'sc_prob_i2.dat')
xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.4, 0.65)
b = 5
xlim = [0,30]
ylim = [0,1]
name = ipt+'sc_prob_i2'
plot2(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

data = np.loadtxt(dcp+'dc_prob_lh.dat')
xlabel = r'$\ln{(E_{i})}$'
ylabel = r'Probability'
lpos = (0.4, 0.7)
b = 1
xlim = [-4.2,1]
ylim = [0,1]
name = ipt+'dc_prob_lh'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

# Initial state 2
data = np.loadtxt(dcp+'dc_prob_i2.dat')
xlabel = r'$\ln{(E_{i})}$'
ylabel = r'Probability'
lpos = (0.35, 0.7)
b = 0.5
xlim = [-3.0,1.0]
ylim = [0,1]
name = ipt+'dc_prob_i2'
plot2(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_prob_lh.dat')
xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.35, 0.6)
b = 2
xlim = [2,30]
ylim = [0,1]
name = ipt+'ec_prob_lh'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_prob_i2.dat')
xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.55, 0.8)
b = 2
xlim = [2,30]
ylim = [0,1]
name = ipt+'ec_prob_i2'
plot2(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

data = np.loadtxt(ecp+'ec_prob_lh2.dat')
lpos = (0.35, 0.6)
name = ipt+'ec_prob_lh2'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

data1 = np.loadtxt(ecp+'ec_prob_lh.dat')
data2 = np.loadtxt(ecp+'ec_prob_lh2.dat')

data[:,0]=np.mean([data1[:,0],data2[:,0]],axis=0)
data[:,1]=np.mean([data1[:,1],data2[:,1]],axis=0)
data[:,2]=np.mean([data1[:,2],data2[:,2]],axis=0)
data[:,3]=np.mean([data1[:,3],data2[:,3]],axis=0)
data[:,4]=np.mean([data1[:,4],data2[:,4]],axis=0)
name = ipt+'ec_prob_lh_mean'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)
```

A.2.4 Plot Transition Probabilities for Parallel Calculations

```
"""
Probabilities Plotting Script.
```

```
3 Author: Daniel Celis Garza
4 """
5
6 import numpy as np
7 import matplotlib
8 matplotlib.use('Agg') # Use .svg format.
9 import matplotlib.pyplot as plt
10 from matplotlib import ticker
11
12 plt.close('all')
13 # Plotting parameters
14 params = {'backend': 'ps',
15           'axes.labelsize': 24,
16           'axes.titlesize': 24,
17           'text.fontsize': 24,
18           'legend.fontsize': 24,
19           'xtick.labelsize': 24,
20           'ytick.labelsize': 24,
21           'text.usetex': True,
22           'font.family': 'serif',
23           'mathtext.fontset': 'custom'}
24 plt.rcParams.update(params)
25
26 scp = '../data/single_avoided_crossing/'
27 dcp = '../data/double_avoided_crossing/'
28 ecp = '../data/extended_coupling/'
29 ipt = '../images/'
30 ext = 'eps'
31
32 def plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext):
33     x, y0, y1, y2, y3 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
34     fig, ax = plt.subplots(figsize=(12,7.42))
35     ax.plot(x, y0,label=r'$ R_{1\leftarrow 1} $', ls = '-', c = 'b', lw = 2.5,
36             marker = 'o', markersize = 10)
37     ax.plot(x, y1,label=r'$ R_{2\leftarrow 1} $', ls = '-.', c = 'g', lw = 2.5,
38             marker = 's', markersize = 10)
39     ax.plot(x, y2,label=r'$ T_{1\leftarrow 1} $', ls = '--', c = 'r', lw = 2.5,
40             marker = 'd', markersize = 10)
41     ax.plot(x, y3,label=r'$ T_{2\leftarrow 1} $', ls = '--', c = 'k', lw = 2.5,
42             dashes=[8, 4, 2, 4, 2, 4], marker = '*', markersize = 10)
43     ax.grid(b=True, which='major', color='0.5',linestyle=':')
44     ax.set_xlabel(xlabel)
45     ax.set_ylabel(ylabel)
46     formatter = ticker.ScalarFormatter(useMathText=True)
47     formatter.set_scientific(True)
48     formatter.set_powerlimits((-1,1))
49     ax.yaxis.set_major_formatter(formatter)
50     plt.legend(bbox_to_anchor=lpos,
51            bbox_transform=plt.gcf().transFigure, frameon=False)
52     ax.xaxis.set_major_locator(ticker.MultipleLocator(base=b))
53     ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=5))
54     ax.yaxis.set_major_locator(ticker.MultipleLocator(base=0.1))
55     ax.set_xlim(xlim)
56     ax.set_ylim(ylim)
57     plt.subplots_adjust(bottom=0.12)
58     plt.savefig(name+'.'+ext, format=ext, dpi=1200)
59
60 # Generating the file with the average value of the two parallel runs for the Single Avoided Crossing
61 data1 = np.loadtxt(scp+'sc_prob1_par.dat')
62 data2 = np.loadtxt(scp+'sc_prob2_par.dat')
63
64 data = np.empty([30,5])
65
66 data[:,0]=np.mean([data1[:,0],data2[:,0]],axis=0)
67 data[:,1]=np.mean([data1[:,1],data2[:,1]],axis=0)
68 data[:,2]=np.mean([data1[:,2],data2[:,2]],axis=0)
69 data[:,3]=np.mean([data1[:,3],data2[:,3]],axis=0)
70 data[:,4]=np.mean([data1[:,4],data2[:,4]],axis=0)
71
72 text_file = open(scp+'sc_prob_parallel.dat', 'w')
73 text_file.write('# 1/5ip, two 7500 MC step runs')
74
75 for i in range(30):
76     text_file.write(str(data[i,0])+' '+str(data[i,1])+' '+str(data[i,2])+' '+
77     str(data[i,3])+' '+str(data[i,4])+'\n')
78 text_file.close()
79
80 xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
81 ylabel = r'Probability'
82 lpos = (0.52, 0.7)
```

```
b = 5
xlim = [0,30]
ylim = [0,1]
name = ipt+'sc_prob_parallel'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

# Generating the file with the average value of the two parallel runs for the Double Avoided Crossing
data1 = np.loadtxt(dcp+'dc_prob1_par.dat')
data2 = np.loadtxt(dcp+'dc_prob2_par.dat')

data = np.empty([26,5])

data[:,0]=np.mean([data1[:,0],data2[:,0]],axis=0)
data[:,1]=np.mean([data1[:,1],data2[:,1]],axis=0)
data[:,2]=np.mean([data1[:,2],data2[:,2]],axis=0)
data[:,3]=np.mean([data1[:,3],data2[:,3]],axis=0)
data[:,4]=np.mean([data1[:,4],data2[:,4]],axis=0)

text_file = open(dcp+'dc_prob_parallel.dat', 'w')
text_file.write('# 1/5ip, two 7500 MC step runs')

for i in range(26):
    text_file.write(str(data[i,0])+' '+str(data[i,1])+' '+str(data[i,2])+' '+
    str(data[i,3])+' '+str(data[i,4])+'\n')
text_file.close()

xlabel = r'$\ln{(E_{i})}$'
ylabel = r'Probability'
lpos = (0.4, 0.7)
b = 1
xlim = [-4.2,1]
ylim = [0,1]
name = ipt+'dc_prob_parallel'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)

# Generating the file with the average value of the two parallel runs for the Extended Coupling
data1 = np.loadtxt(ecp+'ec_prob1_par.dat')
data2 = np.loadtxt(ecp+'ec_prob2_par.dat')
data3 = np.loadtxt(ecp+'ec_prob3_par.dat')
data4 = np.loadtxt(ecp+'ec_prob4_par.dat')

data = np.empty([29,5])

data[:,0]=np.mean([data1[:,0],data2[:,0],data3[:,0],data4[:,0]],axis=0)
data[:,1]=np.mean([data1[:,1],data2[:,1],data3[:,1],data4[:,1]],axis=0)
data[:,2]=np.mean([data1[:,2],data2[:,2],data3[:,2],data4[:,2]],axis=0)
data[:,3]=np.mean([data1[:,3],data2[:,3],data3[:,3],data4[:,3]],axis=0)
data[:,4]=np.mean([data1[:,4],data2[:,4],data3[:,4],data4[:,4]],axis=0)

text_file = open(ecp+'ec_prob_parallel.dat', 'w')
text_file.write('# 1/5ip, four 3750 MC step runs')

for i in range(26):
    text_file.write(str(data[i,0])+' '+str(data[i,1])+' '+str(data[i,2])+' '+
    str(data[i,3])+' '+str(data[i,4])+'\n')
text_file.close()

xlabel = r'Initial Nuclear Momentum ($P_{i}$)'
ylabel = r'Probability'
lpos = (0.35, 0.6)
b = 2
xlim = [2,30]
ylim = [0,1]
name = ipt+'ec_prob_parallel'
plot(data,xlabel,ylabel,lpos,b,xlim,ylim,name,ext)
```

A.2.5 MPI Data Generator Script

```
import numpy as np

data = np.array([0,0,0,0],dtype=float)
text_file = open('mpi_dat.dat', 'w')
for i in np.arange(1,9):
    counter = 0
```

```
    for j in np.array([10,50,100,500]):
        data[counter] = np.loadtxt(str(i)+'_core'+str(j)+'.dat')
        counter += 1
    text_file.write(str(i)+' '+str(data[0])+' '+str(data[1])+' '+str(data[2])+' '+str(data[3])+'\n')
#        for item in np.vstack(data[counter]):
#            print(item)
text_file.close()
```

A.2.6 Plot MPI Data Script

```
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import ticker

plt.close('all')
params = {'backend': 'ps',
          'axes.labelsize': 24,
          'axes.titlesize': 24,
          'text.fontsize': 24,
          'legend.fontsize': 24,
          'xtick.labelsize': 24,
          'ytick.labelsize': 24,
          'text.usetex': True,
          'font.family': 'serif',
          'mathtext.fontset': 'custom'}
plt.rcParams.update(params)

dpt = '../mpi_tests/'
ipt = '../images/'

data = np.loadtxt(dpt+'mpi_dat.dat')
x, y0, y1, y2, y3 = data[:,0], data[:,1], data[:,2], data[:,3], data[:,4]
fig, ax = plt.subplots(figsize=(9.19,14.87))
ax.plot(x, y3,label=r'500 MC reps', ls = '--', c = 'b', lw = 2.5, dashes=[8, 4, 2, 4],
        marker='o', markersize=8)
ax.plot(x, y2,label=r'100 MC reps', ls = '--', c = 'g', lw = 2.5, marker='s', markersize=8)
ax.plot(x, y1,label=r'50 MC reps', ls = '--', c = 'r', lw = 2.5, dashes=[8, 4, 2, 4, 2, 4],
        marker='D', markersize=8)
ax.plot(x, y0,label=r'10 MC reps', ls = '-', c = 'k', lw = 2.5, marker='^', markersize=8)
ax.grid(b=True, which='major', color='0.5',linestyle=':')
ax.set_xlabel(r'Number of Cores')
ax.set_ylabel(r'Computational Time (s)')
formatter = ticker.ScalarFormatter(useMathText=True)
formatter.set_scientific(True)
formatter.set_powerlimits((-1,1))
ax.yaxis.set_major_formatter(formatter)
ax.legend(loc=0)
ax.set_yscale('log')
ax.set_yticks([10, 15, 20, 50, 60, 70, 80, 90, 110, 130,150,170,195,520,600,680,770,870,1000])
ax.get_yaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())
ax.xaxis.set_major_locator(ticker.MultipleLocator(base=1))
ax.xaxis.set_minor_locator(ticker.AutoMinorLocator(n=4))
ax.set_xlim([1,8])
plt.legend(bbox_to_anchor=(0.0, 0.32),loc=2,borderaxespad=0.,frameon=False,fancybox=True,framealpha=1)
plt.subplots_adjust(left=0.15)
plt.savefig(ipt+'mpi.eps', format='eps', dpi=1200)
```

Appendix B

Proofs

B.1 Transformation From Action-Angle to Cartesian Coordiantes

Starting from eq. (B.1),

$$H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{n},\boldsymbol{q}) = \frac{\boldsymbol{P}^2}{2\mu} + \sum_{k=1}^{F} n_k H_{kk}(\boldsymbol{R}) \tag{B.1a}$$

$$+ 2\sum_{k<k'=1}^{F} \sqrt{(n_k+\gamma)(n_{k'}+\gamma)} \times \cos(q_k - q_{k'}) H_{kk'}(\boldsymbol{R}) \ ,$$

$$x_k = \sqrt{2(n_k+\gamma)}\cos(q_k) \tag{B.1b}$$

$$p_k = -\sqrt{2(n_k+\gamma)}\sin(q_k) \tag{B.1c}$$

$$n_k = \frac{1}{2}p_k^2 + \frac{1}{2}x_k^2 - \gamma \ , \tag{B.1d}$$

We first turn to the trigonometric term in eq. (B.1a). We identify the trigonometric identity,

$$\cos(\alpha-\beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) \ , \tag{B.2}$$

and expand the angle-difference in the summation and substitute eqs. (B.1b) and (B.1c),

$$2\sum_{k<k'=1}^{F} \sqrt{(n_k+\gamma)(n_{k'}+\gamma)}(\cos(q_k)\cos(q_{k'}) + \sin(q_k)\sin(q_{k'}))H_{kk'} \tag{B.3a}$$

$$= \sum_{k<k'=1}^{F} \left\{ \begin{array}{c} \sqrt{2(n_k+\gamma)}\cos(q_k)\cdot\sqrt{2(n_{k'}+\gamma)}\cos(q_{k'}) + \\ \left(-\sqrt{2(n_k+\gamma)}\sin(q_k)\right)\cdot\left(-\sqrt{2(n_{k'}+\gamma)}\sin(q_{k'})\right) \end{array} \right\} H_{kk'} \tag{B.3b}$$

$$= \sum_{k<k'=1}^{F} (x_k x_{k'} + p_k p_{k'}) H_{kk'} \tag{B.3c}$$

Which means we now have:

$$H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{n},\boldsymbol{q}) = \frac{\boldsymbol{P}^2}{2\mu} + \sum_{k=1}^{F} n_k H_{kk}(\boldsymbol{R}) + \sum_{k<k'=1}^{F} (p_k p_{k'} + x_k x_{k'}) H_{kk'} \ . \tag{B.4}$$

We then turn our attention to the middle term,

$$\sum_{k=1}^{F} n_k H_{kk}(\boldsymbol{R}) \ . \tag{B.5}$$

In order to have conservation of probability we have the normalisation condition,

$$\sum_{k=1}^{F} n_k = 1 \ . \tag{B.6}$$

We then define the mean of the trace of the diabatic matrix $\bar{H}$,

$$\bar{H} = \frac{1}{F} \sum_{k=1}^{F} H_{kk} \ . \tag{B.7}$$

This means we can write,

$$\sum_{k=1}^{F} n_k H_{kk} \to \bar{H} + \sum_{k=1}^{F} n_k (H_{kk} - \bar{H}) \ . \tag{B.8}$$

We can verify that eq. (B.8) is true by expanding the summation and using the normalisation condition in eq. (B.6),

$$\begin{aligned} \sum_{k=1}^{F} n_k H_{kk} = \bar{H} + \sum_{k=1}^{F} n_k (H_{kk} - \bar{H}) \Rightarrow \bar{H} + \sum_{k=1}^{F} n_k H_{kk} - \sum_{k=1}^{F} n_k \bar{H} \\ \Rightarrow \bar{H} + \sum_{k=1}^{F} n_k H_{kk} - \underbrace{(n_1 + n_2 + \cdots + n_F)}_{=1} \cdot \bar{H} \Rightarrow \sum_{k=1}^{F} n_k H_{kk} \ . \end{aligned} \tag{B.9}$$

Now we recall eqs. (B.7) and (B.8),

$$\sum_{k=1}^{F} n_k (H_{kk} - \bar{H}) = \sum_{k=1}^{F} n_k \left(H_{kk} - \frac{1}{F} \sum_{k=1}^{F} H_{kk}\right) \ . \tag{B.10}$$

By expanding the right hand side we find,

$$\begin{aligned} & n_1 \left[H_{11} - \frac{1}{F} \left(H_{11} + H_{22} + \cdots + H_{FF} \right) \right] + \\ & n_2 \left[H_{22} - \frac{1}{F} \left(H_{11} + H_{22} + \cdots + H_{FF} \right) \right] + \cdots + \\ & n_F \left[H_{FF} - \frac{1}{F} \left(H_{11} + H_{22} + \cdots + H_{FF} \right) \right] \ . \end{aligned} \tag{B.11a}$$

Which can be rearranged to,

$$\frac{1}{F}\left\{\begin{array}{l} H_{11}\left[(F-1)n_1 - n_2 - \cdots - n_F\right] + \\ H_{22}\left[-n_1 + (F-1)n_2 - \cdots - n_F\right] + \cdots + \\ H_{FF}\left[-n_1 - n_2 - \cdots + (F-1)n_F\right] \end{array}\right\} . \tag{B.12}$$

We then take a look at a summation with a different structure,

$$\frac{1}{F}\sum_{k<k'=1}^{F}(n_k - n_{k'})(H_{kk} - H_{k'k'}) , \tag{B.13}$$

where the term $k < k' = 1$ means that k starts at 1 and is always smaller than k'. Table B.1 shows the summation's behaviour.

Table B.1: Behaviour of $\sum_{k<k'=1}^{F} kk'$.

Iteration	Value					
	1	2	3	...	$F-1$	F
1	k	k'	k'	...	k'	k'
2	-	k	k'	...	k'	k'
3	-	-	k	...	k'	k'
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$F-1$	-	-	-	...	k	k'

Expanding eq. (B.13) according to table B.1 we find,

$$\frac{1}{F}\left\{\begin{array}{l} (n_1 - n_2)(H_{11} - H_{22}) + (n_1 - n_3)(H_{11} - H_{33}) + \cdots + (n_1 - n_F)(H_{11} - H_{FF}) + \\ (n_2 - n_3)(H_{22} - H_{33}) + (n_2 - n_4)(H_{22} - H_{44}) + \cdots + (n_2 - n_F)(H_{22} - H_{FF}) + \cdots + \\ (n_{F-1} - n_F)(H_{(F-1)(F-1)} - H_{FF}) \end{array}\right\} . \tag{B.14}$$

Which can be rearranged as eq. (B.15)

$$\frac{1}{F}\left\{\begin{array}{l} \left[(F-1)n_1 H_{11} - n_2 H_{11} - \cdots - n_F H_{11}\right] + \\ \left[-n_1 H_{22} + (F-1)n_2 H_{22} - \cdots - n_F H_{22}\right] + \cdots + \\ \left[-n_1 H_{FF} - n_2 H_{FF} - \cdots + (F-1)n_F H_{FF}\right] \end{array}\right\} , \tag{B.15}$$

and simplified as,

$$\frac{1}{F}\left\{\begin{array}{l} H_{11}\left[(F-1)n_1 - n_2 - \cdots - n_F\right] + \\ H_{22}\left[-n_1 + (F-1)n_2 - \cdots - n_F\right] + \cdots + \\ H_{FF}\left[-n_1 - n_2 - \cdots + (F-1)n_F\right] \end{array}\right\} . \tag{B.16}$$

Which is exactly the same as eq. (B.12), and therefore,

$$\frac{1}{F}\sum_{k}^{F} n_k(H_{kk}-\bar{H}) = \frac{1}{F}\sum_{k<k'=1}^{F}(n_k-n_{k'})(H_{kk}-H_{k'k'})\ . \tag{B.17}$$

Which means our Hamiltonian can be written as,

$$\begin{aligned} H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{p},\boldsymbol{x}) = \frac{\boldsymbol{P}^2}{2\mu} + \bar{H}(\boldsymbol{R}) \\ + \sum_{k<k'=1}^{F}\left\{\begin{array}{r} \frac{1}{F}(H_{kk}(\boldsymbol{R})-H_{k'k'}(\boldsymbol{R}))\cdot(n_k-n_{k'}) \\ +H_{kk'}(\boldsymbol{R})\cdot(p_kp_{k'}+x_kx_{k'}) \end{array}\right\}\ , \end{aligned} \tag{B.18}$$

and using eq. (B.1d), eq. (B.18) can be fully expressed in cartesian coordinates,

$$\begin{aligned} H(\boldsymbol{P},\boldsymbol{R},\boldsymbol{p},\boldsymbol{x}) = \frac{\boldsymbol{P}^2}{2\mu} + \bar{H}(\boldsymbol{R}) \\ + \sum_{k<k'=1}^{F}\left\{\begin{array}{r} \frac{1}{F}(H_{kk}(\boldsymbol{R})-H_{k'k'}(\boldsymbol{R}))\cdot\left(\frac{1}{2}p_k^2+\frac{1}{2}x_k^2-\frac{1}{2}p_{k'}^2-\frac{1}{2}x_{k'}^2\right) \\ +H_{kk'}(\boldsymbol{R})\cdot(p_kp_{k'}+x_kx_{k'}) \end{array}\right\}\ . \end{aligned} \tag{B.19}$$

$\boxed{QED}$

Bibliography

[1] S. J. Cotton and W. H. Miller, "Symmetrical windowing for quantum states in quasi-classical trajectory simulations: Application to electronically non-adiabatic processes," *The Journal of Chemical Physics*, vol. 139, no. 23, p. 234112, 2013.

[2] J. C. Tully, "Molecular dynamics with electronic transitions," *The Journal of Chemical Physics*, vol. 93, no. 2, pp. 1061–1071, 1990.

[3] G. Stock, "A semiclassical self-consistent-field approach to dissipative dynamics: The spin–boson problem," *The Journal of Chemical Physics*, vol. 103, no. 4, pp. 1561–1573, 1995.

[4] C. J. Cramer, *Essentials of computational chemistry: theories and models*. John Wiley & Sons, 2013.

[5] M. E. Sheehan and P. N. Sharratt, "Molecular dynamics methodology for the study of the solvent effects on a concentrated diels-alder reaction and the separation of the post-reaction mixture," *Computers & Chemical Engineering*, vol. 22, pp. S27–S33, 1998.

[6] S. Mandal, M. Moudgil, and S. K. Mandal, "Rational drug design," *European Journal of Pharmacology*, vol. 625, no. 1, pp. 90–100, 2009.

[7] B. K. Shoichet, S. L. McGovern, B. Wei, and J. J. Irwin, "Lead discovery using molecular docking," *Current Opinion in Chemical Biology*, vol. 6, no. 4, pp. 439–446, 2002.

[8] J. F. Blake, "Integrating cheminformatic analysis in combinatorial chemistry," *Current Opinion in Chemical Biology*, vol. 8, no. 4, pp. 407–411, 2004.

[9] G. Schneider and H.-J. Böhm, "Virtual screening and fast automated docking methods," *Drug Discovery Today*, vol. 7, no. 1, pp. 64–70, 2002.

[10] T. N. Doman, S. L. McGovern, B. J. Witherbee, T. P. Kasten, R. Kurumbail, W. C. Stallings, D. T. Connolly, and B. K. Shoichet, "Molecular docking and high-throughput screening for novel inhibitors of protein tyrosine phosphatase-1b," *Journal of Medicinal Chemistry*, vol. 45, no. 11, pp. 2213–2221, 2002.

[11] A. M. Paiva, D. E. Vanderwall, J. S. Blanchard, J. W. Kozarich, J. M. Williamson, and T. M. Kelly, "Inhibitors of dihydrodipicolinate reductase, a

key enzyme of the diaminopimelate pathway of mycobacterium tuberculosis," *Biochimica et Biophysica Acta (BBA)-Protein Structure and Molecular Enzymology*, vol. 1545, no. 1, pp. 67–77, 2001.

[12] M. S. Lesney, "Nature's pharmaceuticals," *Today's Chemist at Work*, vol. 7, pp. 27–32, 2004.

[13] W. L. Jorgensen, "The many roles of computation in drug discovery," *Science*, vol. 303, no. 5665, pp. 1813–1818, 2004.

[14] Stanford University, "Folding@Home," http://folding.stanford.edu/home/.

[15] Harvard University, "the Harvard Clean Energy Project," http://cleanenergy.molecularspace.org/.

[16] E. Lewars, *Computational chemistry: introduction to the theory and applications of molecular and quantum mechanics*. Springer, 2010.

[17] M. Karplus and J. A. McCammon, "Molecular dynamics simulations of biomolecules," *Nature Structural & Molecular Biology*, vol. 9, no. 9, pp. 646–652, 2002.

[18] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel *et al.*, "Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit," *Bioinformatics*, p. btt055, 2013.

[19] F. Aquilante, L. De Vico, N. Ferré, G. Ghigo, P.-å. Malmqvist, P. Neogrády, T. B. Pedersen, M. Pitoňák, M. Reiher, B. O. Roos *et al.*, "Molcas 7: The next generation," *Journal of Computational Chemistry*, vol. 31, no. 1, pp. 224–247, 2010.

[20] M. H. Holmes, *Introduction to perturbation methods*. Springer, 2012, vol. 20.

[21] MIT, "Big O," http://web.mit.edu/16.070/www/lecture/big_o.pdf, Sept. 2014.

[22] I. Kassal, S. P. Jordan, P. J. Love, M. Mohseni, and A. Aspuru-Guzik, "Polynomial-time quantum algorithm for the simulation of chemical dynamics," *Proceedings of the National Academy of Sciences*, vol. 105, no. 48, pp. 18 681–18 686, 2008.

[23] D. J. Dean, J. Gour, G. Hagen, M. Hjorth-Jensen, K. Kowalski, T. Papenbrock, P. Piecuch, and M. Włoch, "Nuclear structure calculations with coupled cluster methods from quantum chemistry," *Nuclear Physics A*, vol. 752, pp. 299–308, 2005.

[24] B. P. Lanyon, J. D. Whitfield, G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri *et al.*, "Towards quantum chemistry on a quantum computer," *Nature Chemistry*, vol. 2, no. 2, pp. 106–111, 2010.

[25] R. Kosloff, "Time-dependent quantum-mechanical methods for molecular dynamics," *The Journal of Physical Chemistry*, vol. 92, no. 8, pp. 2087–2100, 1988.

[26] R. A. Kendall, E. Aprà, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha *et al.*, "High performance computational chemistry: An overview of nwchem a distributed parallel application," *Computer Physics Communications*, vol. 128, no. 1, pp. 260–283, 2000.

[27] Q.-C. Inc., "Q-Chem," http://www.q-chem.com/qchem-website/manual_4-1.html.

[28] T. H. Dunning Jr, "Gaussian basis sets for use in correlated molecular calculations. i. the atoms boron through neon and hydrogen," *The Journal of Chemical Physics*, vol. 90, no. 2, pp. 1007–1023, 1989.

[29] E. R. Davidson and D. Feller, "Basis set selection for molecular calculations," *Chemical Reviews*, vol. 86, no. 4, pp. 681–696, 1986.

[30] G. Karlström, R. Lindh, P.-Å. Malmqvist, B. O. Roos, U. Ryde, V. Veryazov, P.-O. Widmark, M. Cossi, B. Schimmelpfennig, P. Neogrady *et al.*, "Molcas: a program package for computational chemistry," *Computational Materials Science*, vol. 28, no. 2, pp. 222–239, 2003.

[31] M. Karplus and J. Kuriyan, "Molecular dynamics and protein function," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 19, pp. 6679–6685, 2005.

[32] S. P. Hunt and P. W. Mantyh, "The molecular dynamics of pain control," *Nature Reviews Neuroscience*, vol. 2, no. 2, pp. 83–91, 2001.

[33] D. Marx and J. Hutter, "Ab initio molecular dynamics: Theory and implementation," *Modern Methods and Algorithms of Quantum Chemistry*, vol. 1, pp. 301–449, 2000.

[34] M. Karplus, R. Porter, and R. Sharma, "Exchange reactions with activation energy. I. Simple barrier potential for (h, h2)," *The Journal of Chemical Physics*, vol. 43, no. 9, pp. 3259–3287, 1965.

[35] S. J. Cotton and W. H. Miller, "Symmetrical windowing for quantum states in quasi-classical trajectory simulations," *The Journal of Physical Chemistry A*, vol. 117, no. 32, pp. 7190–7194, 2013.

[36] J. F. Stanton and J. Gauß, "CFOUR," http://cfour.de/.