Benchmarking Ab Initio Computational Methods for the Quantitative Prediction of Sunlight-Driven Pollutant Degradation in Aquatic Environments

An Honors Thesis for the Department of Chemistry

By Kasidet (Jing) Trerayapiwat*

-

Bowdoin College, 2016

Abstract

Understanding the changes in molecular electronic structure following the absorption of light is a fundamental challenge for the goal of predicting photochemical rates and mechanisms. Proposed here is a systematic benchmarking method to evaluate accuracy of a model to quantitatively predict photo-degradation of small organic molecules in aquatic environments. An overview of underlying computational theories relevant to understanding sunlight-driven electronic processes in organic pollutants is presented. To evaluate the optimum size of solvent sphere, molecular Dynamics and Time Dependent Density Functional Theory (MD-TD-DFT) calculations of an aniline molecule in different numbers of water molecules using CAM-B3LYP functional yielded excited state energy and oscillator strength values, which were compared with data from experimental absorption spectra. For the first time, a statistical method of comparing experimental and theoretical data is proposed. Underlying Gaussian functions of absorption spectra were deconvoluted and integrated to calculate experimental oscillator strengths. A Matlab code written by Soren Eustis was utilized to decluster MD-TD-DFT results. The model with 256 water molecules was decided to give the most accurate results with optimized computational cost and accuracy. MD-TD-DFT calculations were then performed on aniline, 3-F-aniline, 4-F-aniline, 3-Cl-aniline, 4-MeOacetophenone, and (1,3)-dimethoxybenzophenone with CAM-B3LYP, PBE0, M06-2X, LCBLYP, and BP86 functionals. BP86 functional was determined to be the best functional.

Contents

1 Introduction

1.1 Environmental Photochemistry And Micropollutants

During the past century, more and more synthetic chemicals have been created for commercial pharmaceuticals and personal care products. Each year, about 300 million tons of organic chemicals are being added into water systems from household septics, waste water facilities, and many other paths.[1] These chemicals have been detected in ng/L to g/L quantity in natural water bodies: rivers, lakes, and the oceans.[2] The amount of chemicals being introduced into the water raises the need to understand the consequences of these compounds. If the water is polluted, eventually animals, human, and other living beings in this world are affected. For example, in 1956, Minamata disease was identified to be caused by consumption of fish bio-accumulated by methylmercury from a nearby chemical factory.[3] While some major toxic chemicals, such as mercury, CCl_4, DDT, etc., have been constantly regulated or banned by governments for their effects on agriculture and the environment, many more chemical wastes have not been studied and regulated to the same degree. Even though some chemicals are approved as safe for daily usage in household products, their impacts after being disposed into water remain largely unknown. These low-concentration chemicals are collectively called micropollutants.[1] The small concentration, large number of different species in water, and difficulty of separating or concentrating them for analysis all retard the effort to understand their effects on in water systems. These are "The Challenge of Micropollutants in Aquatic Systems".[1] Since each individual species of micropollutant is not classified as harmful, major concerns of introducing them to water bodies are their reactions in water. In water with exposure to sunlight, photoreaction can occur as photons from the sun energize organic pollutants from ground state to excited state. In rivers close to discharge areas, photolysis has been confirmed to have direct connection to the observed attenuation in concentration of micropollutants from upstream to downstream.[5–8] Observable decrease in concentrations means a significant portion of the pollutants undergo photo-reactions in exposure to sunlight. Eustis et al. carried out computational studies of triclosan in excited states.[9] Triclosan goes through photolysis under sunlight to form Dioxins and

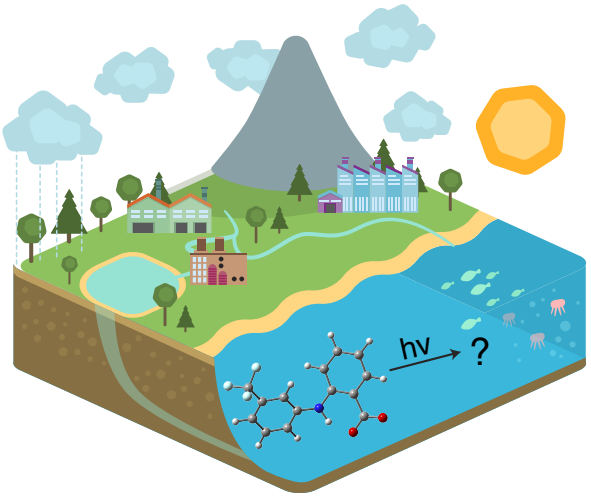


Figure 1: Environmental fate of micropollutants in water. Under sunlight, molecules can undergo unknown photo-reactions to form unknown products.[4]

PCBs, well-known carcinogens.[10] This is but one of many examples of large-scale, unintended consequences as a direct result of persistent, low-level concentrations of pollutants in aquatic ecosystems[11]. One important step in triclosan's degradation process to its 4 major products is detachment of ortho-Cl from triclosan ring under 300 nm light. Confirmed by combining both experimental and computational results in the paper, ortho-chlorine photo-dissociates from triclosan through reductive cleavage pathway through biradical anion. Excited state bond energy of the breaking C-Cl reveals significantly smaller activation energy in reductive cleavage pathway compared to that of through homolytic mechanism. In this case, experimental method provides evidence of products from photo-reaction. Computational method provides reasons and insights into the microscopic world of atoms and electrons. Combing and analyzing results the two methods give a better understanding of photo-reactions in aquatic solutions.

1.2 Photo-excitation and UV-VIS Spectrum

Understanding each step in the photochemically driven reaction is critical to properly evaluate the impact of a pollutant's environmental fate. The first, and most important step in this process is light absorption of ground state molecules to form chemically activated excited state molecules. A molecule in its ground state (S_0) can absorb a photon to its first singlet excited state (S_1), and then relax to its first triplet excited state (T_1) as illustrated in Figure 2. Experimentally, studying absorption of photons in chemical compounds is done through UV-VIS absorption spectroscopy, followed by analysis of molar absorptivity and wavelength. In contrast to common misconception, a value of molar absorptivity is not a characteristic property of a compound. Molar absorptivity as a function of wavelength is highly dependent on the resolution used in the measurement.[13] Solvent-solute interactions also play a role in changing the excited state energies.[14–21] One particular study done by Eilmes suggests that the effects solvent molecules have on solute can be separated into two main parts: solvent absorption peak shift and solvent broadening .[22] Two sets of Molecular Dynamics and Time Dependent Density Functional Theory (MD-TD-DFT) calculations were done controlling either solute or solvent molecules. The results were analyzed to study how either solvent or solute geometry variation af-

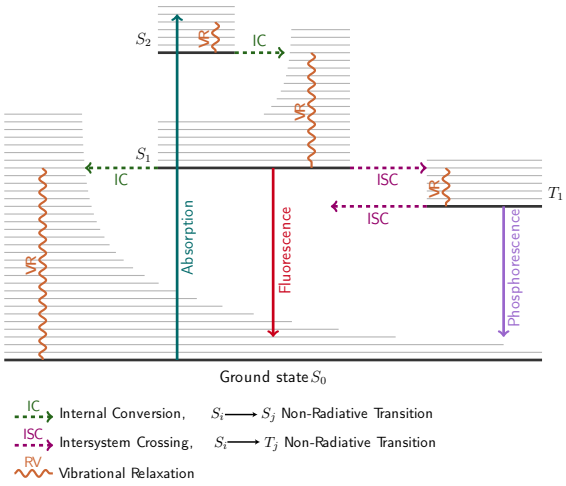


Figure 2: Jablonski diagram.[12] A possible pathway of photo-excitation starts at ground state to singlet excited state, then triplet excited states.

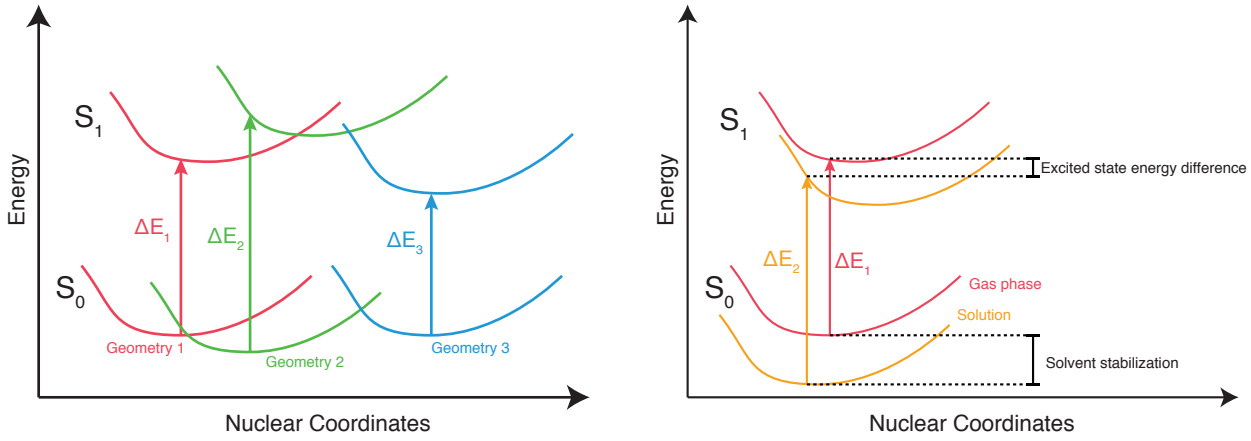


Figure 3: Illustration of solvent effects on potential energy surfaces of a solute. a) different ground state geometries give different potential energy surfaces for ground and excited states. Excited states energy calculation then gives different results due to this solute geometric fluctuation. The result is solvent peak broadening around the mean. b) Solvent stabilization of the ground state. Ground state geometry has lower potential energy surface as a result from dipole interaction from solvent molecules. Generally, the excited state geometries are not as well equilibrated by the ground state solute and solvent geometries.

fected the first excited state energy. First, the solute molecule were not static in the solution and geometry fluctuation creates a broad range of ground state Potential Energy Surfaces (PES) (see Figure 3a). Different starting ground states yield different predictions for the solute's excited states. This contributes to absorption peak broadening compared with an absorption in gas phase. In addition to solvent broadening, temperature broadening due to Boltzmann distribution of molecular energies around the global minimum energy as described in Molecular Dynamics calculation also contribute to absorption peak broadening.

Secondly, dipole moments from the solvent stabilize the ground state geometry lowering the potential energy surface. Since electrons move significantly faster than the nucleus, excited state geometries are less likely to be stabilized. Unequal lowering of the two potential energy surfaces causes blueshift in excited state energy prediction (see Figure 3b). In gas phase, simulation of UV-VIS spectrum can be done more easily due to a lack of solvent shift and solvent broadening. In solution, sharp and high-intensity gas-phase peaks become broadened and blueshifted. Each individual absorption

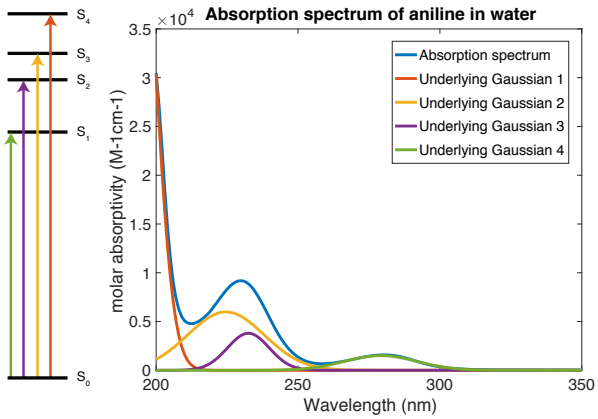


Figure 4: Absorption spectrum of aniline in water showing underlying electronic transitions. Underlying Gaussian components, fitted manually using Fityk, refer to 4 electronic transitions as shown in electronic transition diagram on the left. Each peak can be used to calculate oscillator strength using equation 16

peak of an electronic transition over a period of wavelength may overlap. A molar absorptivity vs excitation energy plot is a linear combination of normal distribution plots from different electronic transition in a compound. Each individual peak combines together into a continuous experimental spectrum as seen in Figure 4. Notice different peak width at different excitation modes. Broadening effects from ground state geometry fluctuation affects their excited states differently. Once molecules are excited, solute molecules exist in the high energy state for a period of time. During that time a molecule can undergo different changes: undergoing a reaction by itself, emitting fluorescence light, or relaxing non-radiatively and de-energize back to the ground state. The rate constant of a reaction by the excited state (k_{rxn}) and the rate of photon absorption (k_{abs}) can determine the likelihood of a certain reaction to occur with respect to the amount of photons absorbed. A quantum yield (Φ)for a photo-reaction can then be calculated using Equation 1:

$$\Phi = \frac{\#\ of\ molecules\ undergoing\ a\ reaction}{\#\ of\ photons\ absorbed} = \frac{k_{rxn}}{k_{abs}} \tag{1}$$

Two main components needs to be determined to evaluate the quantum yield: the reaction rate constant and the light absorption rate constant. Reaction rate can be determined computationally from calculating two potential energy surfaces of the reactant and the product and solving for the conical intersection, where the two surfaces meet.

In Eustis lab, Nathan D. Ricke '14 worked on applying computational model to estimate quantum yields of photo-excited small organic molecules in the singlet excited states undergoing Intersystem Crossing (IC) from to the first triplet excited state using Landau-Zener model and outlining a possible method using Fermi's golden rule.[23] Landau-Zener model calculates the probability of IC using spin-orbit coupling constant at minimum crossing geometry, energy gradient, and crossing velocity. The rate is then calculated from probability from Arrhenius equation with activation energy being the energy difference between the singlet excited state energy and the conical intersection. Quantum yields of aniline and benzene calculated using this theory were compared with the experimental values. Qualitative agreement with the experiments were observed and discussed. An alternative method is Fermi's golden rule, which calculates rates of transition from one energy state to another from electronic couplings,

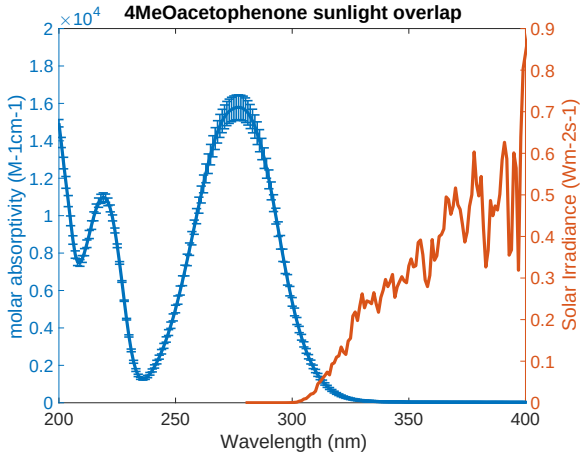


Figure 5: Overlap between absorption spectrum of 4-MeOacetophenone and solar radiation (ASTM G173) showing at which wavelength photons from sunlight can be absorbed to energize the molecule to the excited states.

Franck-Condon factors, and solvent broadening factor. However, the calculation could not be done due to memory shortage caused by the number of terms generated at higher vibrational excitations. Nevertheless, Landau-Zener model can be used to calculate k_{rxn} of IC, the numerator term in Equation 1.

Since reactions of micropollutants in aquatic environment are the focus of this study, the main photon source for the photo-reactions is the sun. Sunlight at different wavelengths goes through the ozone layer, but most light in the UV range is absorbed by the ozone layer.[24] Some of the remaining light photons pass through the atmosphere before hitting the earth surface at a 90 degree. Some disperse in the atmosphere (commonly referred to as sky radiation[25]) before hitting the earth surface at different angles. Light intensity in terms of Solar irradiance (W) can be measured experimentally or looked up on a reference table: ASTM G173-03 Reference Spectra based on conditions considered average in most states in the US and tilt angle representative of the contiguous US states.[26] In water body, photon intensity decreases as a function of water depth and its attenuation characterized by the beam attenuation coefficient (α) and depth (z), both measured experimentally. These physical quantities of sunlight can be combined with the molar absorptivity (ε) of a species in water to give rate of light absorption (k_{abs}) defined by Equation 2:

$$k_{abs}(\lambda) = \frac{W(\lambda) \cdot \varepsilon(\lambda)}{z \cdot \alpha(\lambda)} \tag{2}$$

The focus of this study is to calculate molar absorptivity or equivalent in order to calculate the photon absorption rate in the aquatic environment. Study of UV-VIS absorption processes combined with the two methods outlined by Nathan Ricke illustrate what further calculations can be done to predict the quantum yield of sunlight-driven pollutant degradation in water.

1.3 Computational Approach

In recent years, high performance computers have become more and more accessible to researchers.[27] The power of High Performance Computing (HPC) extends the frontier of rigorous computational calculations to solve problems which a decade before were prohibited by the high computational cost (time and resources). To put "high cost" in perspective, wall clock time required to optimize geometry and vibration states of aniline in gas phase using different theories are shown in table 1. Computational time in the table was generated to provide guidance to understanding computation cost barrier faced when using higher-level theories. Theories are sorted in order of increasing computational time. Due to its accuracy, double excitation Coupled Cluster (CCSD)[28,29] is considered the hallmark of computational chemistry. Unfortunately, computational time required for CCSD limits its application to only 10-20 molecules; optimization of 14-atom aniline requires 15 computational days. Compared with normal Hartree Fock, CCSD takes 2000 times longer. For larger

systems, CCSD becomes impossible to be implemented. With larger number of CPU and more efficient CPU, wall clock time can be faster than computational time. If basis sets size doubles, the time required can be

Table 1: Comparing time required to optimize aniline's geometry and vibrational states and ground state energy using Hartree Fock (HF), DFT, MP2, and CCSD level theory in Gaussian '09[30] to demonstrate how computational time and accuracy varies with different theories. All calculation were done using 6-31G(d) basis sets on a 32-processor node and 10 GB connection.

Theory	Scaling factor	Time	Time if basis sets double in size
HF	N^4	10 mins	2.7 hours
DFT	N^3	20 mins	2.7 hours
MP2	N^5	1 hours	32 hours
CCSD	N^6	15 days	2.6 years

calculated from the scaling factor (the last column in Table 1). It becomes clear that CCSD cannot be used for any calculation in a large system. None of the theories here scale linearly, which should reflect certain difficulties when the system becomes larger. MP2 also starts to take longer compared with HF or DFT. Computational chemists can now implement more rigorous ab initio (from the first principles) methods. As more difficult problems are explored, there is a trend to incorporate both experimental and theoretical methods together for analysis because they have different advantages and disadvantages. For example when studying the photo-degradation of micropollutants, computational approach allows for a priori prediction of photo-products and their impact on the environment. Experimental approach is more limited to a posteriori study of negative impact micropollutants have on water system. Experimental chemists face challenges to study individual species of micropollutants due to complex experiment planning. In natural waters, many species of micropollutants interfere with each other setting limits on what can be measured experimentally in a complex natural system. Experimental methods excel when used to study an observable natural phenomena and its consequences. While theoretical methods are great for studying simple system with good accuracy, they have gained attentions from scientists just in the past decades thanks to burgeoning of growth in computating speed. The computational theories are still in a stage of development as more and more researchers have more access to faster computers. They have become very accurate at calculating small and simple systems. However, ab initio theories remain incomplete and inaccurate for larger

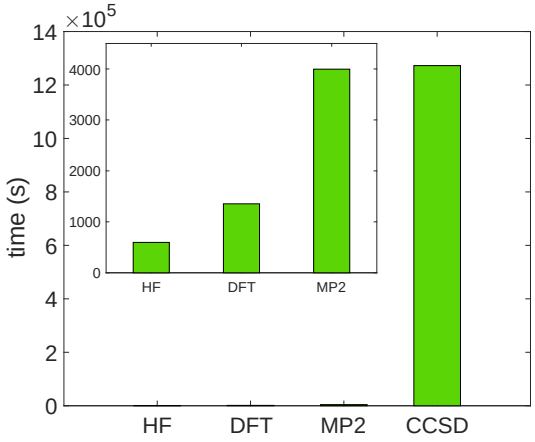


Figure 6: Visualization of table 2. Due to their low computational costs compared with CCSD, time values of the other three theories are plotted in a box on the left hand corner.

and more complex systems. There is still a significant trade-off between cost and accuracy; the question is how much accuracy is needed? There are many factors that increase accuracy and computational cost: Theories, basis sets, solvent models...

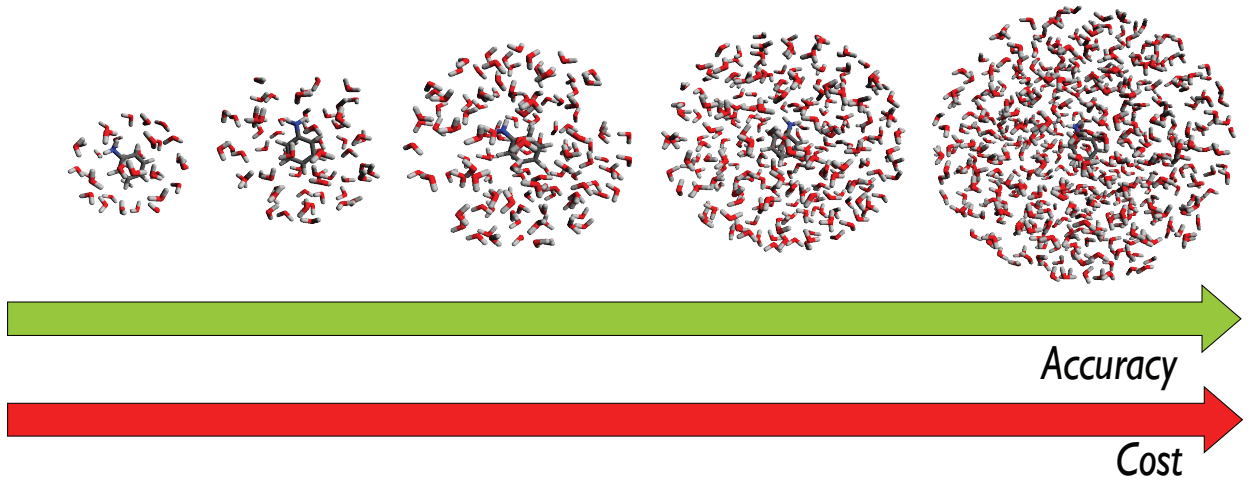


Figure 7: Explicit 2^n water model (n = 5-9). The more water, the more complete solute-solvent model but the slower the calculation.

The objective of this project is to find a computational model with the best trade of between accuracy and computational cost to simulate excited states energy and probability of excitation of small organic molecules in water compared with experimental results. A work procedure will be developed to systematically evaluate the accuracy of different models. If successful, this procedure will be applicable to other larger molecules of different chemical properties. Criteria of being the best model include good accuracy when compared with the experimental data, reasonable computational cost, and applicability to different models. The best model to calculate the excited states will lead to rate constants of photon absorption, quantum yields, and general prediction of photo-transformation pathways of small organic pollutants.

2 Background

2.1 Hartree-Fock Theory

Schrödinger equation is an equation describing the relationship between the energy of system (E) and the wave function (Ψ) of a wave-particle. It is written in time-independent form:

$$H\Psi = E\Psi \tag{3}$$

$$H = -\frac{\hbar^2}{2m}\nabla^2 + V \tag{4}$$

with H is the Hamiltonian operator, which describe what the energy of the system is controlled, written here in italics instead of with a circumflex mark over the operator (in physics and computational chemistry, H is the convention more commonly used). The left operator is the kinetic energy operator with Laplace operator written using the del symbol. For simple external potential energy conditions like particle in a box or harmonic oscillator, the wave function can be deduced from the Schrödinger equation and its energy calculated. More information about Schrödinger equation can be found in any elementary quantum mechanics books.[31,32] For a system with more than two or more electrons, there is no direct way to solve the Schrödinger equation for an analytic wave function. Variational principle can be used to find the most "correct" numerical wave function that satisfies an eigenfunction in lieu of the Hamiltonian operator in a certain theories. With a trial wave function (ϕ_{trial}), energy calculated from the Hamiltonian operator is always higher than energy calculated using the true wave function (ψ_{real}):

$$\langle\psi^*_{real}|H|\psi_{real}\rangle \leq \langle\phi^*_{trial}|H|\phi_{trail}\rangle \tag{5}$$

According to this principle, one can keep modifying parameters in the trial wave function until the calculated energy converges to the lowest value before concluding that the trial wave function is the best representation of the real wave function. The lowest energy calculated is by no means the "correct" value. Theories make assumptions and those assumptions limit how accurate the lowest energy of a theoretical calculation can achieve. Another useful assumption widely used in quantum chemistry is Linear Combination of Atomic Orbitals (LCAO) which allows *atomic* orbital wave functions (χ_{n}) to be summed to create a *molecular* wave function (ϕ_{i}):

$$\phi_i = a_{i1}\chi_1 + a_{i2}\chi_2 + a_{i3}\chi_3 + +a_{i4}\chi_4 + ... \tag{6}$$

Wave function theories mostly use this assumption. One fundamental theory, on which many other theories are base, in computational chemistry is Hartree-Fock Theory (HF). HF theory make assumptions about wave functions and the nature of quantum mechanical interactions in the atomic systems, including LCAO, single-electron wave functions decoupling, and non-relativistic electrons, to arrive at Fock operator (F) for a single-electron wave function:

$$F_i = h_i + \sum_{j}^{N}(J_i - K_j) \tag{7}$$

with h being the kinetic and electron-nucleus Coulomb potential energy of an electron, J electron-electron Coulomb interactions, K electron-electron exchange (a term for Pauil Exclusion Principle). Eigenvalues of F are matrices with Lagrange coefficients. These coefficients can be used to calculate energies of each one-electron wave functions before being added back together to calculate total energy of the system. Iterations of guessing a trial wave function through LCAO coefficient adjustment, operating F on each single-electron wave functions, and energy calculations are done until the system with the lowest energy is achieved. Limitations of HF are in its assumptions. First, electrons are coupled so single-electron wave functions cannot truly represent one electron without taking into account electron-electron correlations. One of many post-HF theories, most notably Møller—Plesset perturbation theory (MP) [33], attempts to add electron-correlation to correct errors introduced by approximations in HF.

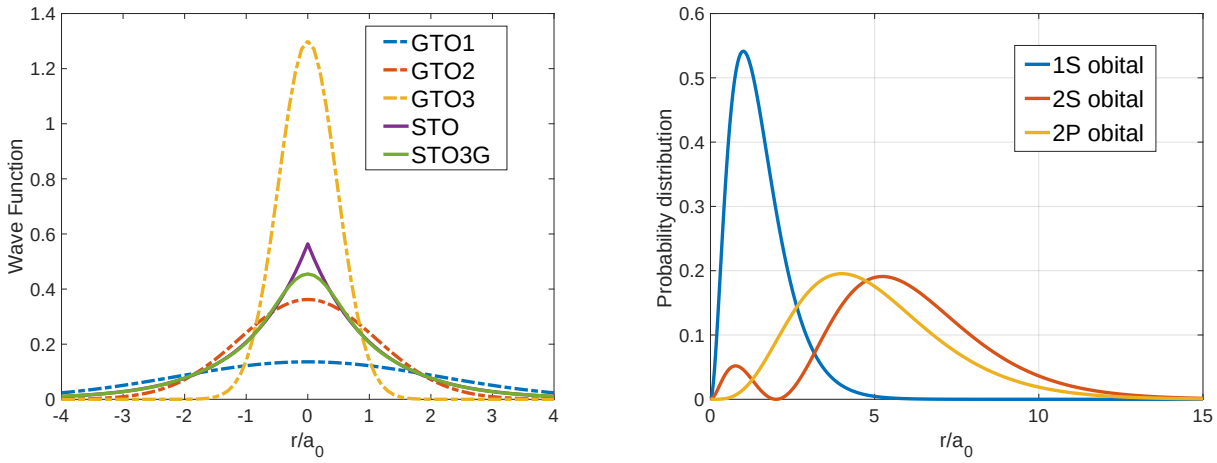
2.2 Basis Sets

To describe electron orbitals in a molecule, wave functions for quantum calculations are modeled using mathematical functions called basis sets. Simple Gaussian-type basis sets are a combination of Gaussian distributions modeling different orbitals in Cartesian coordinates. They were designed by John Pople to be optimized for speed. [34] Smaller basis sets have fewer basis functions per atom. In addition to having higher functions per atom, larger basis sets may also include polarized and diffuse basis sets. Equation 6 can be rewritten to include two 1S orbitals of two hydrogen atoms in hydrogen gas:

$$\phi_{H_2} = a_1\chi_{H_A}^{1S} + a_2\chi_{H_B}^{1S} \tag{8}$$

While this wave function is adequate to describe ground state hydrogen gas, it will certainly fail to describe the excited state which requires orbitals in the second electron shell for the electrons in the excited states. Therefore, more orbitals can be added to better describe the molecular orbitals:

$$\phi_{H_2} = a_1\chi_{H_A}^{1S} + a_2\chi_{H_A}^{2S} + a_3\chi_{H_A}^{2P} + a_4\chi_{H_B}^{1S} + a_5\chi_{H_B}^{2S} + a_6\chi_{H_B}^{2P} \tag{9}$$



(a) **(b)**

Figure 8: Visualization of basis sets. a)plots of STO and STO-3G with its Gaussian components. All three Gaussian plots with 0.44, 0.54, and 0.15 mixing coefficients are added to STO describe atomic orbitals better. First, there is no more discontinuity in the first derivative of the wave function as in STO. Gaussian functions are also easier to integrate compare with exponential functions. Data replotted with permission from Magalhães[35]. Copyright 2014 American Chemical Society. b) probability density of different atomic orbitals. Diffuse functions can be added to light and heavy atoms to better describe more diffuse orbitals when electrons move further away from the nucleus. In this figure, 2S and 2P can increase the distance an electron wave function describe with expanded probability distribution.

The atomic orbitals can also be described by analytical mathematical expressions controlled by numerical parameters like Gaussian distributions. For example, in describing the molecular orbitals in aniline molecule, there are different level Gaussian basis set to choose from: STO-3G, 3-21G, 6-31+G(d), ..., 6-311++G(2d,p). STO[36] is a Slater-type minimal basis set designed for fast calculation. It is the solution to 1S Hydrogen atom using only an exponential function. In contrast, STO-3G mixes STO with 3 more Gaussian functions to describe the wave function more accurately (see figure 8a). 3-21G is also a small model (cheap and simple). It contains 3 Gaussian functions for inner shell orbitals, 2 and 1 different level of Gaussian functions describing valence electrons. 6-311++(2d,p) is more complete but also more expensive. It contains 6 Gaussian functions for inner shell orbitals, 3, 1, and 1 different level of Gaussian functions describing valence electrons. In addition, ++ means it adds two large s orbitals and one large p orbitals *diffuse* functions. This is important in calculating reactions with electron being further away from the nucleus such as electron addition, electronic transition, or charge transfer. Finally, (2d,p) means heavier atoms also contain two d orbitals polarized to the P orbitals and hydrogen atoms p functions to s orbital. Polarized basis sets becomes important when describing physical properties involving electron excitations.

Another one of the most well-known basis sets is the Dunning's correlation-consistent polarized Core and Valence (Double/Triple...) Zeta (cc-pVXZ) type basis sets. They are basis sets in spherical coordinates

that contain variationally optimized coefficients for electron correlations (in contrast to Pople's which was optimized for Hartree Fock). Dunning's cc-pVXZ basis sets may include a large number of polarized basis sets based on what level of Zeta is used. For example, cc-pV6Z describes a second-row atom using 140 basis sets.[37] Diffuse basis sets can be added to cc-pVXZ with aug- keyword in front of the basis sets. In comparing among theoretical calculation, aug-cc-pVQZ (ACCQ) can be used as theoretical benchmark.[38,39] ACCQ's computational cost is however prohibitive for solute-solvent model with Bowdoin HPC.

Table 2: Demonstrating how time required to optimize hydrogen gas geometry and vibrational states and ground state energy using different basis sets in Gaussian '09. All calculation were done with HF level of theory on a 24-processor node and 64 GB of RAM. This table underscores trade-off between accuracy and cost using aug-cc-pvQz as the theoretical benchmark

Basis Set	Time (seconds)	Energy (Eh)	R (A)
3-21G	134	-1.1230	0.735
6-31G	133	-1.1268	0.730
6-311G	137	-1.1280	0.732
6-31G(d,p)	154	-1.1313	0.732
6-311G(d,p)	153	-1.1325	0.735
6-311G(d,2p)	163	-1.1330	0.734
6-311G(d,3p)	161	-1.1331	0.734
6-311G(d,3pd)	194	-1.1331	0.734
aug-cc-pvQz	3250	-1.1335	0.734

To put computational time required by different basis sets in perspective, hydrogen gas geometry can be optimized using different Gaussian-type basis sets and aug-cc-pv6z. This computational demonstration illustrates the main points addressed below about how size of different basis sets affect time and accuracy. As basis sets size increased, computational time increased and energy converged closer and closer to that of the theoretical benchmark. 6-311G(d,2p) is the best basis sets for geometry optimization because it used 20% less time to calculate compared with 6-311G(2d,p), but with the same degree of accuracy. It is worth noting that depending on the type of calculations and the size of the systems, different basis sets should be chosen to best balance accuracy and cost. Adding (d,p) polarized basis sets helped get more accurate H-H radius both in 6-31G to 6-31G(d,p) and 6-311G to 6-311G(d,p). However, within Gaussian-type basis sets group, changes in time as more basis sets added was not as drastic as moving from 6-311G(d,3pd) to ACCQ which takes 15 times longer time to run. When the system becomes larger, the calculation time scaled would become longer. Increasing the size did not always increased accuracy. Note that the radius calculated using 6-311G(d,p) converged to 0.734 Angstroms, close to theoretical benchmark at 0.735 Angstroms. Increasing the size of the basis sets after did very little to improve the value. Different types of calculation then would be more suitable with different basis sets: 6-311G(d,p) for geometry optimization and 6-311G(d,2p) for energy calculation. In calculating excited state energies of organic molecules[39,40], while having cheaper cost, an average-sized

basis set 6-311++G(2d,p)[41] performs better than aug-cc-pVDZ (ACCD)[41]. For example, transition energies calculated of CN molecule as calculated by ACCD deviates 1117-1669 cm^{-1} from experimental value while those by 6-311++G(2d,p) only deviate 220-470 cm^{-1}.[39]

2.3 Density Functional Theory

In theory, Density Functional Theory (DFT) does not require making assumptions about electron-electron decoupling because it does not encounter problems with degrees of freedom as wave-function-based methods do. It therefore has potential to be more exact compared to HF or post-HF wave function methods. Despite its low costs, DFT's accuracy in certain applications has been accepted and acclaimed.[42,43] Wave-function-based methods locate each particles in wave functions with 3 parameters on cartesian coordinates.

$$\Psi(x_1, y_1, z_1, x_2, y_2...) \tag{10}$$

In multi-particle systems, computational cost rises exponentially with the number of particles. For a large solute-solvent system, the calculation cost becomes expensive. DFT uses electron density function with only 3 Cartesian parameters. Despite having fewer parameters, density functions also contain data about total number of electrons, position of nuclei, and atomic number; all required to calculate the Hamiltonian operator.

$$\rho(x, y, z) \tag{11}$$

Energy of a density function is not calculated based on HF or other wave function methods. Historically, DFT had large errors and limited usage in computational chemistry because there is no analytical or numerical way to solve for electron density functions. In wave function based methods, variational principles governs that energy calculated from the operating Hamiltonian on a candidate wave function gives higher energy than on the correct wave functions. By adjusting the wave function, until its energy is minimized, a wave function as a solution to that Hamiltonian operator can be solved for numerically. Two important theorems developed by Pierre Hohenberg and Walter Kohn allow DFT to be used more broadly in computational chemistry community by providing a way to calculate the energy via variational principle.[44] The first theorem states that only ground state electron density is required to construct the external potential function and hence the Hamiltonian. This is important because potential for excited state wave functions can also be calculated. The second theorem, is the variational principle for DFT allowing to change parameters in the density

function to get lowest energy through wave function and Hamiltonian calculation.

$$\left\langle \Psi^*_{cand(DFT)} \left| H \right| \Psi_{cand(DFT)} \right\rangle = E_{cand(DFT)} \geq E_0 \tag{12}$$

The problem with DFT as described is variational principle calculation on the wave functions because it requires solving the Schrödinger's equation to obtain the wave function and make assumptions/approximations as discussed above. To solve this, Kohn-Sham Self-consistent Field (Kohn-Shan SCF), developed in 1965, made use of the two theorems by breaking down electron density to non-interacting single-electron density functions.[45] Energy functionals can then operate on linear combination of these electron functionals to give the energy of the system. Functionals are functions which operate on another function. Instead of using the Hamiltonian operator and a wave function to calculate the energy (Equation 3), Kohn-Shan SCF calculates the energy of the system directly from electron density function using different functionals. For example, the Kohn-Sham one-electron operator as presented in Cramer's Essentials of Computational Chemistry[37]:

$$h_i^{KS} = -\frac{1}{2}\nabla_i^2 - \sum_k^{Nuclei} \frac{Z_k}{|r_i - r'|} + \int \frac{\rho(r')}{|r_i - r'|} dr' + V_{xc} \tag{13}$$

with the familiar looking first two terms (see Equation 4). The last term on the right hand side of the equation refers to exchange-correlation functional; it includes quantum mechanical exchange, electron correlation, and correction in kinetic and potential terms. Exchange interaction in fermions like electrons follows Pauil Exclusion Principle, constraining electrons from being described by the same quantum numbers. Correlation is an electron-electron interaction term ignored in HF. In modern DFT functionals, experimentally fitted parameters are used to describe these energy functional components.[37] Different functionals have different fitted parameters and their accuracy dependent on the system and physical properties being calculated. To find the best group of functionals for calculating excited state energies, there needs to be a benchmark specifically on excited state energy calculation on similar systems and electronic states. In modern functionals, a functional of electron density gradient function is added to the energy functionals.These functionals are gradient corrected and the method called generalized gradient approximation (GGA). Pure GGA functionals is known to underestimate system energy[46–49] or excited state energy[39,50]. In addition, a percent of contribution calculated from HF can be mixed in with the pure GGA results to create GGA-HF hybrid functionals. According to Hellmann-Feynman theorem (Equation 14), an actual exchange-correlation energy can be described as an energy plot over extent of correlation (λ), as shown with green shaded area in Figure 9 recreated here based on figures in two literature sources.[37,51] Area of box A and B respectively refers to pure HF exchange energy and pure GGA exchange-correlation energy. Since the energy as a function of extent

of correlation is not known, experimentally fitted value (z) is needed to find the integral using Equation 15. Adding HF exchange energy reduces overestimation of energy calculated from GGA functionals.

$$E_{xc} = E_{xc} = \int_0^1 \langle \Psi(\lambda) | V_{xc} | \Psi(\lambda) \rangle \, d\lambda \tag{14}$$

$$E_{xc} = E_x^{HF} + z(E_{xc}^{GGA} - E_x^{HF}) \tag{15}$$

Functionals with HF component are called hybrid functionals. Some of the more widely-used hybrid functionals include: B3LYP, CAM-B3LYP[52], M06-2X[53], and PBE0[54]. New functionals can also be created from mixing one functional for exchange and one for correlation. For example, BP86 is a combination of B88 for exchange and P86 for correlation. Unlike basis sets, which can become more complete as number of basis sets per atom increases, there is no 'full,' or complete, functional to use. Among all current theories to calculate the excited states, Time-Dependent Density Functional Theory (TD-DFT) is the most promising with its high accuracy when used with appropriate functionals and low computational cost[55]. TD-DFT calculate the electron density response to excitation photon modeled as an external perturbing field varying with time.[56]

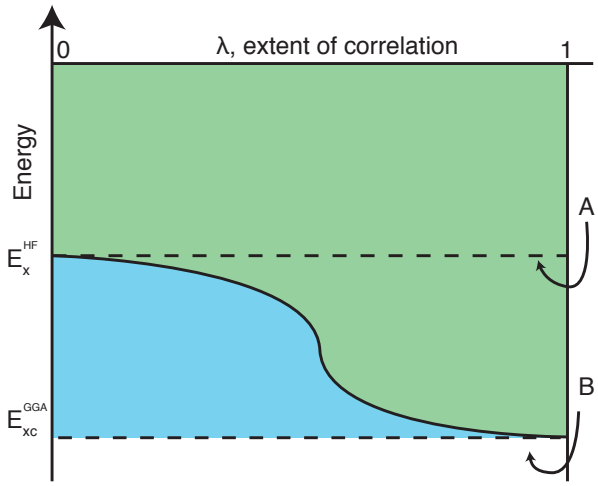


Figure 9: Illustration of hybrid exchange-correlation term from HF and DFTrecreated here based on figures in two literature sources.[37,51]. Box A drawn by the first dotted line is an integral of exchange-only energy calculated from HF from λvalue 0 to 1. Box B drawn by the second dotted line is the same integral but of exchange-correlation energy calculated from pure-GGA DFT. The value of actual exchange-correlation energy (shaded green) is likely to be described by the value between area of box A and B.

Despite its under-performance in certain systems like organic dyes[57] and sulfur organic compounds[58], studies suggest limitations can be overcome if appropriate funcitonals are used.[49,59] According to one study of 0-0 transition in radicals, the two best-performing DFT functionals are CAM-B3LYP and M06-2X. Pure GGA functional BP86[60,61] and long range corrected GGA LCBLYP[60,62,63] are also attractive to have a diverse group of functionals.

2.4 Solvent Models

Despite the recent advent of growth in computer speed and the burgeoning interest in incorporating computational models to further understand the natural world, large systems such as solvation models remain

a big challenge.[64] In describing the solvents, one can either apply bulk solvent effect on the solute through different potentials without specifying the solvent molecules (implicit solvent model) or one can explicitly place the solvent molecules around the solute molecule (explicit solvent model). When calculating excited states energy of a solute, solute-solvent interactions will contribute to modeling solvent effects on absorption spectra as previously discussed (Figure 3a and 3b). In the past, implicit solvation models were frequently implemented because they gave acceptable results while maintains low computational cost. In contrast to explicit model, interactions between each individual solvent molecule and the solute molecule are not calculated. The most commonly used implicit model is the Polarizable Continuum Model (PCM).[65–67] Instead of explicitly handling each solvent molecule quantum mechanically, PCM expresses their bulk effects on solute molecule in means of dielectric continuum field surrounding molecule of interest. Dielectric field can polarize and affect the wave function of the solute system. Its disadvantage is its lack of dynamic solvent contribution on excited states properties.[59] Implicit solvent models also neglect hydrogen-bonding as they assume implicit implementation in dispersion forces and electrostatics.[68] In molecules with hydrogen bonding, implicit solvent model fail at quantum mechanical level to fully model solute-solvent interactions. Having more rigorous calculations, explicit solvent models, require more computational cost compared with implicit water models. Cost for treating a solute molecule in a large solvent sphere with full quantum mechanics can become prohibitive depending on basis sets, theory, the number of explicit solvent molecules (see Table 1 and Figure 7). One recent notable model, Effective Fragment Potentials (EFP), can be used to model explicit solvent molecules with non-bonded van der Waals interactions, hydrogen bonding. It implements Coulomb interactions, polarization, and exchange repulsion without high computational expense of explicit models.[69,70] Implementing EFP solvent model, TD-DFT can be used to accurately calculate excited state energy of acetone in water.[70] In incorporating the solvent model, there are also two different methods: static or dynamics. In static model, geometry optimization of solute molecule is carried out, then followed by calculation of excited state energies. This static ground state molecule however does not accurately represent solute in water.[71] As discussed earlier, the system molecules are dynamic. The difference in solute geometries can broaden the solute's excited state energies in solvent. Instead, Molecular Dynamics (MD) of solute and EFP solvent fragments can be used to obtain a range of equilibrated structures at a specific temperature (see Figure 12a). Mark Gordon et al. averaged the calculated energies of each excited state to arrive at a final excited states energy.[71]

2.5 Molecular Dynamics

Canonical Ensemble Molecular Dynamics (NVT) [72] provides a pool of equilibrated ground state geometries under constant number of particles, volume, and room temperature, 298K (see Appendix). Starting with a higher-energy initial geometry, a system relaxes to lower potential energy state until it reaches the dynamic equilibrium. However, there is no accepted measure of equilibrium. One study by Chodera suggests looking at how density of liquid argon changes over time and using variance-bias trade off to find equilibrium period. [73] By eliminating non-equilibrated geometries, the rest of the equilibrated geometries become "equilibrated" samples. As requirements for the equilibrium determination are relaxed, more "equilibrated" samples can be used for further calculations, the variance of the value calculated decreases. However, error from including non-equilibrated geometries (bias) increases as more samples, which has not really reached the equilibrium enter the pool. The best trade-off were determined to be a cut off-time when the ratio of sampling size to estimated statistical inefficiency is maximized. For a large system like organic solvent-solute model, this algorithm cuts off too much of the sampling size so it follows that its estimate of statistical inefficiency is not accurate. According to the study, the accuracy of cut-off should be questioned if the whole sample time is not at least an order of magnitude larger than cut-off time value.

2.6 Comparing Experimental And Theoretical Calculation Results

As discussed above, experimental molar absorptivity is not a characteristic of a chemical property; the value depends on solvent broadening and measurement resolution. Molar absorptivity at a single wavelength value(ε, $M^{-1}cm^{-1}$), with a value range of 0 to 50000 $M^{-1}cm^{-1}$, may qualitatively imply probability of transition. The larger the molar absorptivity, the larger the absorptivity, the more photons absorbed. Molar absorptivity does not have an upper limit. Results obtained from theoretical calculations are composed of excited states energy (E, nm) and oscillator strength (no unit). [74] Oscillator strength, with range from 0 to 1, is a measure of how probable a specific electronic transition can occur. There is no solvent broadening involved in calculating static oscillator strength from a geometry, therefore theoretical oscillator strength cannot be compared with molar absorptivity directly. Oscillator strength tells the degree of allowness of transition between two electronic states. Mathematically, oscillator strength can be calculated from integrating molar absorptivity over a range of wavenumber within an electronic transition (see equation 16). [75]

$$f = 4.3 \times 10^{-9} \int_{\widetilde{\nu}_a}^{\widetilde{\nu}_b} \varepsilon(\widetilde{\nu}) d\widetilde{\nu} \tag{16}$$

3 Methods

3.1 Overview

Systematic approach to developing computational model is outlined in this section. Python [76] scripts were used to facilitate some of the steps (see Appendix). Aniline was chosen as the initial test solute due to its hydrogen bonding capability, its small size, and extensive previous experimental and theoretical work on aniline within the Eustis lab. A successful model of aniline was expected to be applicable even to larger molecules of similar class, organic and aromatic. Starting molecule geometries were obtained from optimizing the molecular geometry of the solute and solvent molecules yielding two optimized geometries for solvent and solute. These two geometry files were then inputed into Packmol program [77] with empirical guess initial solvent radius and number of water molecules to create starting geometry of solvated solute. MD-TD-DFT using the geometry from Packmol was performed using GAMESS quantum computational package. All MD run used 6-311++G(2d,p) basis sets, calculating velocity vector every 1 fs while calculating other physical properties displaying every 10 fs at bath temp = 298 K. All TD-DFT calculations use 6-31+G(d,p) basis set. Water molecules were modeled explicitly using EFP1 and positioned initially using Packmol. Criterion for MD equilibrium is that the average change of systems potential energy over time crosses zero. After equilibrium has been reached, a 10 ps of equilibrated geometries are used in calculating excited state energies using TD-DFT in GAMESS. Matlab [78] was used to deconvolute experimental spectra and theoretical data cluster. The flowchart for finding the best model is outlined below in Figure 10. First, the optimal number of water molecules in the model was evaluated by using different numbers of water molecules surrounding aniline with CAM-B3LYP functional. The results were compared with experimental data to evaluate the accuracy of the model. The model which contained the lowest number of water, but still provided the most accurate results compared with experimental data is chosen, then applied to other molecules with different functionals before compared again to experimental data.

3.2 Computational

3.2.1 Facilitating Each Steps Using Python Scripts

Python scripts play an important role in both data collection and smoothing the process between each computational steps. In order to automatically generate input files and cultivate output data from output files, many python scripts were written from scratch (see Appendix). Since each script is specific to each calculation task, there is a limited number of scripts available on the internet (virtually none for this project) Log files obtained from GAMESS contains both valuable theoretical data and useless text strings. For

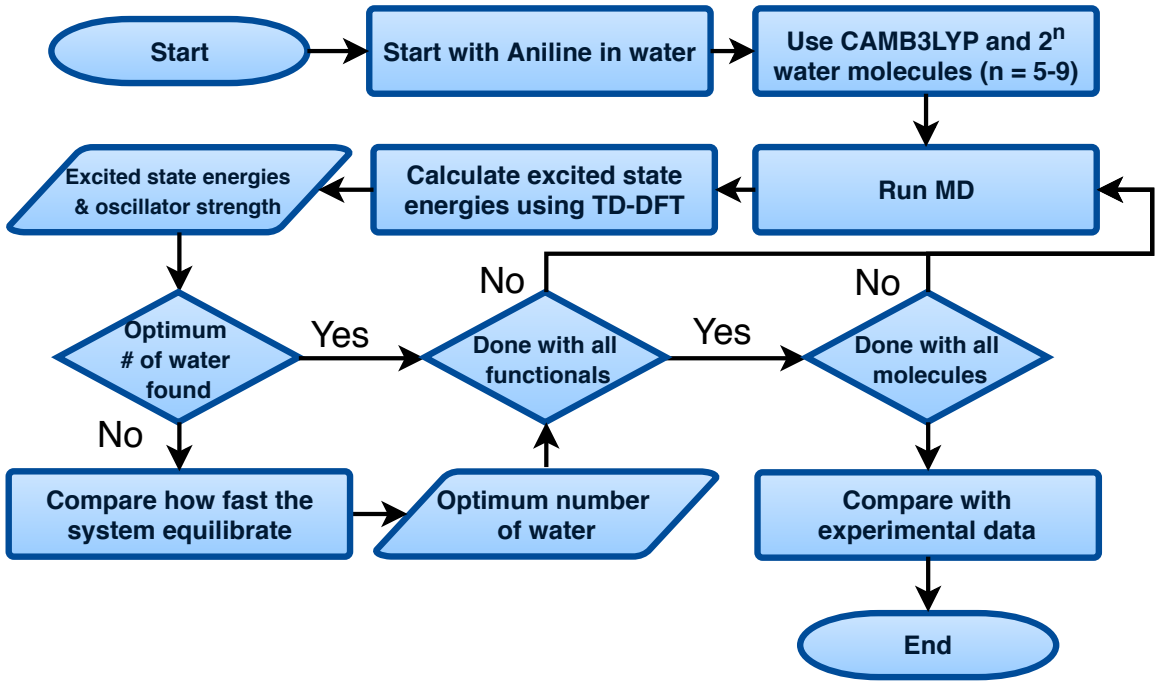


Figure 10: Flowchart of work in this project. Python scripts were utilized to facilitate certain steps. Section and page number of important steps are listed here: reason for "Start with aniline in water," Section 3.1 on page 17; procedure for "Use CAMB3LYP and 2^n water molecules," Section 3.2.2 on page 19; results and discussion of "Run MD" and "Calculate excited state energies using TDDFT" Section 4.1 on page 22; "Compare how fast the system equilibrate," Section 4.2 on page 27; "Compare with experimental data," Section 4.3 on page 29;

example, even though WebMO[79] automatically generates last geometry calculated from MD run, extracting a geometry from each MD step requires one to manually open the log file, copy-paste, reformat each geometry into an input file for TD-DFT calculation one geometry at a time. The python script postMDDataPull2.py is designed to pull thousands of geometries and generate GAMESS input files for TD-DFT energy calculation within seconds. In some cases, using Python scripts was required. When running MD of 512 water molecules, the size of the log file became so large (15 GB). The text files that size are unsupported by any regular graphical interface text editor. Extra care were taken when designing these scripts to deal with large data to make sure fast speed is maintained and results consistent when applied to different files. Generating these python scripts will also allow unified program to be developed in order to automate the whole project without any manual input.

3.2.2 Preparing Starting Geometry Using Packmol

As in many theoretical calculation, an initial guess is required. In MD, initial geometries of the solute and solvent were required to start MD calculations in GAMESS. This guess geometry was calculated using Packmol by inputting optimized geometries of solute and solvent calculated from using DFT with CAM-B3LYP functional and 6-311G(2d,p) basis sets. A sample of Packmol input file for solvating aniline in 16 water molecules is displayed below. The program does not return system geometry unless solvent molecules can fit in smallest, user-specified, radius without exceeding steric strain controlled by tolerance value (line 25). The value, 2.5 Angstrom, controlled the smallest distance atoms need to be far away from each other (line 6). If molecules could fit, solvent radius is increased slightly until a geometry was obtained. Notice this input command asks for spherical solvent sphere, which is convenient to use when implementing SSBP. If periodic boundary condition is to be used, line 25 should be changed to "inside cube 0. 0. 0. 5." with a cube of dimension 5 x 5 x 5 Angstrom^3.

```
1  #
2  # aniline solvated by 16 waters
3  #
4
5  # define minimum distance between atoms to 2.5 Angstrom
6  tolerance 2.5
7  # number of loops for fitting
8  nloop 10000
9  # number of trial per loop
10 maxiter 100
11
12 filetype xyz
13 # specify the solute
14 structure aniline.xyz
15   number 1
16   # set center of mass at (0,0,0) without any rotation
17   fixed 0. 0. 0. 0. 0. 0.
18   centerofmass
19 end structure
20
21 # specify the solute
22 structure h2o.xyz
23   number 16
24   # set center of solvent sphere at (0,0,0) with 5 Angstrom radius
25   inside sphere 0. 0. 0. 5.
26 end structure
27
28 output aniline16.xyz
```

3.2.3 Computational Models

Packmol was used to place water molecules around solute molecules to create starting geometry. GAMESS computational chemistry software was used in calculating MD and excited state energies. Fityk program[80] was used to fit Gaussian under absorption spectrum (Figure 4), and theoretical data (Figure 12b). In order to

accurately calculate the excited energies, the 6-311++G(2d,p) basis sets was chosen to run vertical excitation energies on a set of equilibrated geometries. It was decided to reduce the basis set in running MD. A smaller basis set 6-31+G(d)[41] was used in order to cut computational cost. In MD calculation, diffuse and polarized basis sets are not as important since electrons do not move as much compared with TDDFT where charge transfer occurs when electrons move from HOMO to LUMO. It is an accepted knowledge that smaller basis sets produces as accurate geometry compared with larger basis sets (see also Table 2). Smaller basis sets also converge more easily giving better accuracy-cost trade off. EFP1 model of water molecules was chosen to implement an explicit solvent model and calculate excited states energies. The appropriate number of water molecules in the model had never been evaluated. Too many number of water molecules require more expensive computational cost. Too few water will not give a full model of solute-solvent interaction. A binary framework was used to test how many water molecules were needed to fully solvate the aniline: 2^n, n=5 to 9 (see Figure 7).

After number of solvent molecules suitable to model solute-solvent for photo-excitations in aniline was decided, MD-TD-DFT excited state energies and oscillator strengths were calculated for 3-F-aniline, 4-F-aniline, 3-Cl-aniline, 4-MeOacetophenone, and (1,3)-dimethoxybenzophenone using that model. Once excited state energies for each system were calculated, the results were compared with experimental spectral data to find which functional performed best at calculating MD-TD-DFT excited states energy. Since CAM-B3LYP is a good functional for calculating ground state energies, equilibrated geometries will be calculated using CAM-B3LYP, and TD-DFT excited states using different functionals. Other functionals are PBE0, M06-2X, BP86, and LCBLYP. Aspreviously discussed, M06-2X was expected to perform well when calculating excited state energies (see Introduction). PBE0 is a generic hybrid functional. BP86 is a pure GGA function (no HF). LCBLYP is a long range corrected functional.

3.3 Experimental

Experimental absorption spectra of 6 small organic molecules (aniline, 3-F-aniline, 4-F-aniline, 3-Cl-aniline, 4-MeOacetophenone, (1,3)-dimethoxybenzophenone) in pH 7 water was collected by Alex Poblete '17 and Holly Rudel '17 over the summer of 2015. Experimental methods as written in a currently submitted but unpublished article[81] is rewritten here in full. The organic analytes were purified via vacuum distillation under 99.998 % nitrogen atmosphere and/or as described in the literature[82]. Stock solutions (1 - 10 mM) were made in purified methanol and stored in amber vials at 4 degrees Celsius. Type I water (AquaSolutions, $\geq 18\ M\Omega^1 cm^{-1}$) was used for all aqueous solutions. Solutions were buffered at pH 7.0 with 10 mM phosphate buffer. Quartz glassware (Starna Cells, USA) was used for all spectroscopic measurements. UV-Vis

absorption measurements were performed using a dual-beam (sample/blank) Cary 400 spectrometer. The instrument is calibrated for energy accuracy (holmium oxide gel, Starna) and transmittance (Dichromate) on a regular basis. scanning from 190 - 400 nm at 1 nm increments at a rate of 50 nm per minute. The slit width was fixed at 1 nm resolution. Blank solutions were prepared from the same 10 mM phosphate buffered Type I water. Instrument correction was performed by inclusion of a blank in each concentration series, and via real-time reference subtraction against an additional blank during each measurement. Each analyte series consisted of a blank and five samples in the concentration range of 5 - 100 µM The concentration range chosen for each analyte was chosen to avoid concentrations approaching the solubility limit and to provide $\varepsilon_{\text{lambda}}$ values ≤ 0.1 absorbance units for the most concentrated sample. Issues with insolubility at pH 7 / 25 degrees C were avoided by examining available solubility data[83], assuring all stock and working solutions are fully stirred at room temperature before transfer.

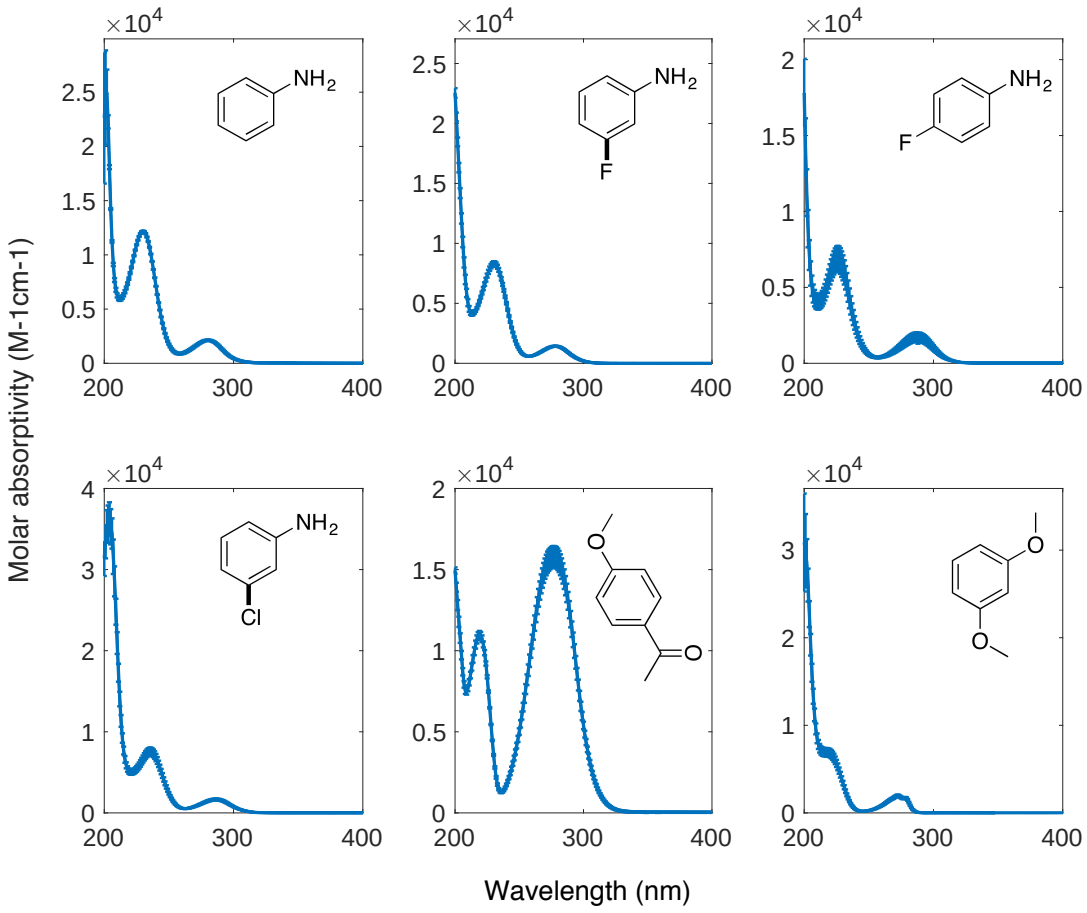
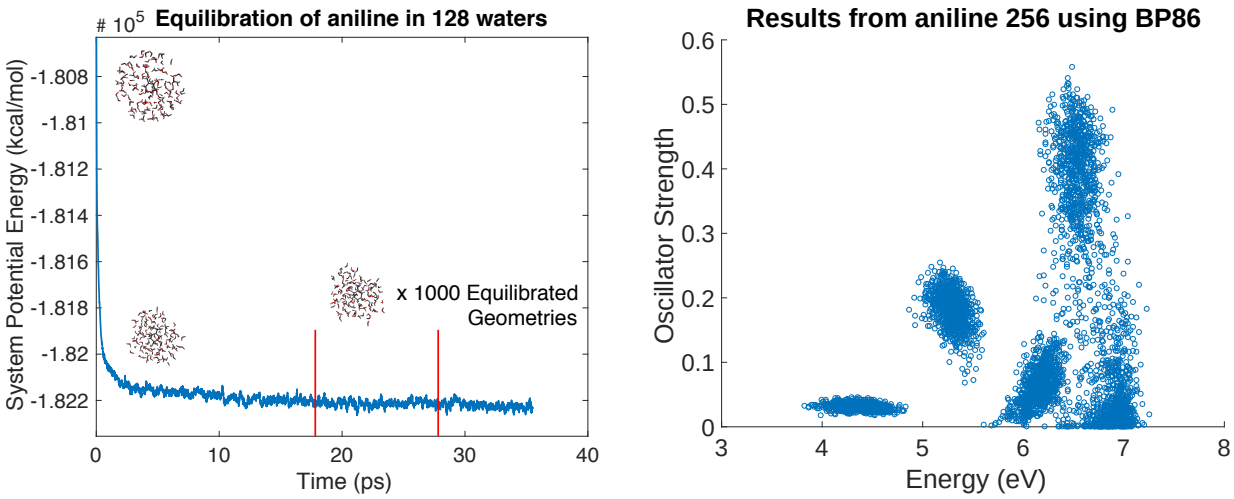


Figure 11: Experimental UV-VIS spectra of 6 organic molecules measured in pH 7 water and 298K collected by Alex Poblete '17 and Holly Rudel '17 over the summer of 2015. Each spectrum represents the fitted slope of six samples with values of c from 0 to 100M, with the Y-error bar representing the standard error or 95% confidence interval.

4 Results And Discussion

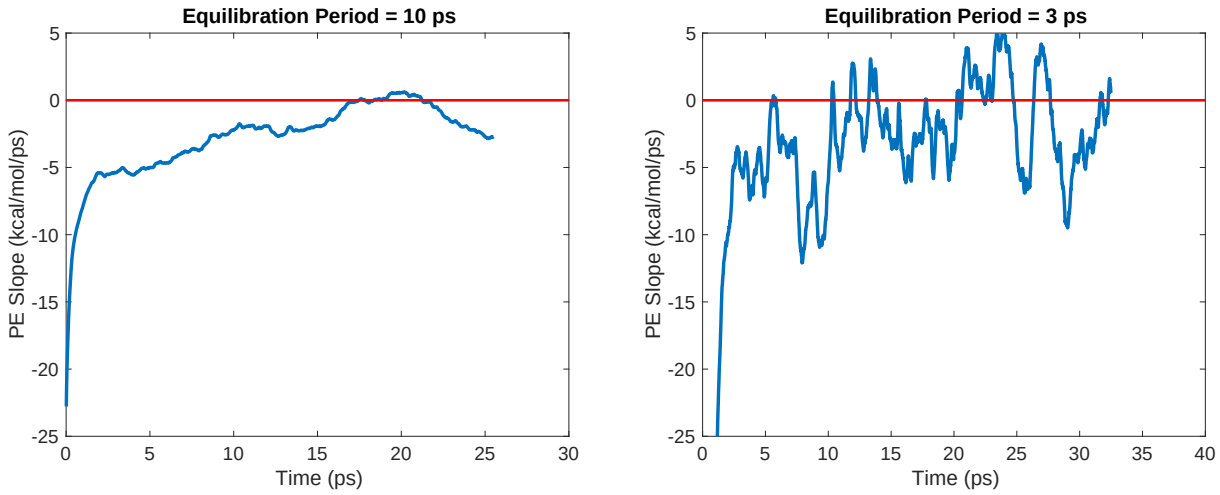
4.1 Results from MD-TD-DFT



(a) **(b)**

Figure 12: MD run and excited state data calculated from TDDFT of equilibrated geometries from MD. a) plot of system's potential energy vs time in MD with geometries shown at t=0, 0.2, 1.78, 2.78 ps. System energy and solvent radius decreased very rapidly in the first 0.2 ps, before converging to the equilibrium value. Fluctuation in energy is strong, around 0.1% but the equilibrium was determined to be from 1.78-2.78 ps (see methods). b) theoretical data from MD-TDDFT calculation using 1000 equilibrium geometries. Roughly five groups of data can be seen here. It would be erroneous however to assume each excited state energy calculated from one geometry comes from the same excitation mode.

To explain procedures and methods used to analyzed the results, MD-TD-DFT result using aniline in 128 water molecules using BP86 functionals was used to as an example case. MD's system potential energy as a function of time was plotted here in Figure 12a. Starting solute-solvent geometry at 0 picoseconds (ps) was obtained from Packmol program. Within 20 ps, system energy dropped 1%. Water sphere radius also appeared to have shrunk from its starting geometry. The decrease in solvent radius was expected. Hydrogen bonding and other intermolecular interactions stabilize the system causing system energy to lower to its equilibrium value. After the first 2 ps, system energy decreased, taking time to slowly equilibrate. In this case, 34.5 ps worth of MD was calculated took 9 days on 32-cpu and 10 Gb connection. Chodera's code[73] was used to calculate the cut-off time to be at 30 ps. The sample size should be at least one order of magnitude larger than the equilibrated one for the assumption made to estimate statistical efficiency used in the code to hold true. Assuming this cut-off point to be accurate, at least 300 ps of MD data would be obtained before deciding to choose use 30 ps as a cut-off point with confident. Projecting this forward, it would take 3 months to calculate MD run with time span larger enough to determine the equilibrium



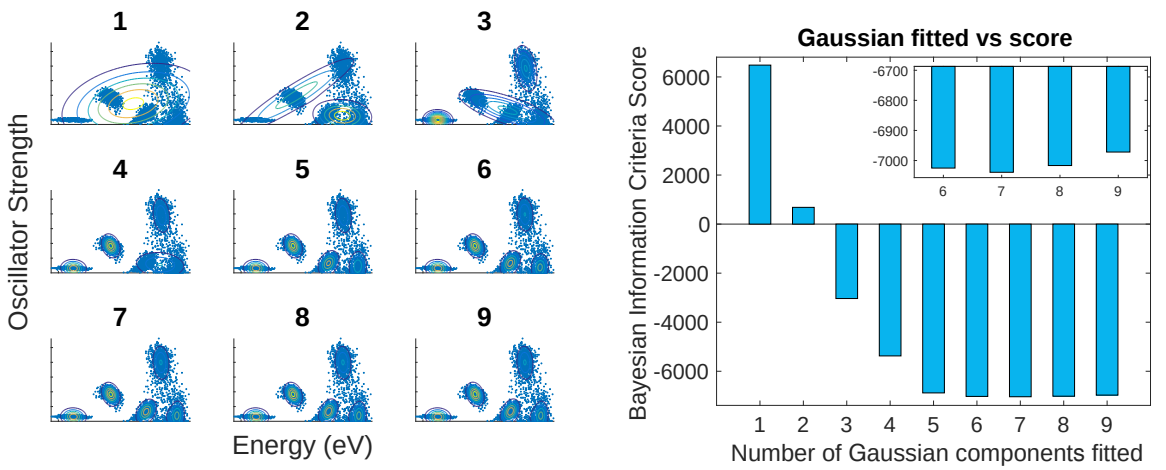
(a) **(b)**

Figure 13: Slope plot of aniline with 128 water BP86 using different equilibrium period. a) 10 ps. b) 3 ps. Notice the higher in slope fluctuation in (b) compared to (a). This is due to smaller sampling size

using Choreda's method. Instead of using Chodera's code, the equilibrium was determined by plotting linear regression slope of system's potential energy over time with an arbitrarily set interval of 10 ps (equilibration period). The consistently low fluctuation indicated the start of equilibrium at 18 ps. Once the system's energy fluctuation reached zero, that interval should be considered at equilibrium. Slope of linear fit for each 10 ps interval starting at time on x-axis was displayed on Figure 13a. When equilibration period was changed to 3 ps (Figure 13b), energy slope reached zero earlier compared to 10-ps. The new equilibrium period yields equilibrium to be established at 6 ps instead of 20 ps. However, reducing equilibration period created larger energy slope fluctuation due to smaller sample size. Systematic criteria of measuring the equilibrium should still be properly explored and studied.

After MD equilibrium was found in the 1.78 to 2.78 ps region, system geometries of that region were used to calculate excited states using TD-DFT. Five excited energy states and its oscillator strengths from the ground state were calculated and shown in Figure 12b. Since pairs of excited state energy and oscillator strength data are collected over a range of equilibrated starting geometries, statistical analysis of raw theoretical data is required. A Matlab code was developed by Soren Eustis using Bayesian probability to fit Gaussian distribution to raw energy-oscillator-strength pair from Figure 12b to Figure 14a. The number of Gaussian plots fitted ranges from 1-9 in each sub-figure. The accepted number of plots was determined and the excited state energies and oscillator strengths given by Bayesian Information Criteria score. The lower the score of a sub-figure, the better fit. According to Figure 14b, 7 Gaussian components fitted to theoretical data was the best. Results of this fit is shown in the next section.

Experimental absorption spectra needed to be deconvoluted into a mixture of Gaussian functions for



(a) **(b)**

Figure 14: Results from Bayesian analysis using a Matlab code developed by Soren Eustis. a) deconvoluted theoretical excited state energies (E, eV) and oscillator strength (f, unitless) with varying number of Gaussian components from 1 to 9. Notice even after 5 apparent clusters were plotted, more Gaussian functions were can still be added. b) Bayesian information criteria score as a function of increasing number of fitted Gaussian functions. The lower the score, the better the fit. In this particular plot, the best number of Gaussian plots is 7.

oscillator strength calculation of each individual electronic transition (see Figure 4). Subsequently, each individual function was integrated numerically over the entire energy range of the function to give the total transition intensity using Equation 16. After calculation of excited state energies and the oscillator strength, computational results were compared with values calculated from experimental UV-VIS spectrum to evaluate accuracy of the models tested. There are currently no studies using full statistical method on quantitative calculation of excited state energies of organic molecule in water. Currently, Matlab code utilizing Bayesian probability as a statistical technique to properly find the Gaussian plots are being developed by professor Soren Eustis from previous work in R[84] with Peter Cohen '18. This allows room for a systematic approach to develop an appropriate computational model that would allow for further understanding of aquatic pollutants. In special cases, more information can be extracted from MD energy plot and system geometries. Two example of this will be discussed in the following sections.

4.1.1 Solvent Boundary Potential

At room temperature, volatile solvent molecules like water can evaporate off the solvent shell during MD calculation. In preliminary runs, MD of aniline with 4 water molecules were calculated using CAM-B3LYP functional. System potential energy shown in Figure 15a as a function of time did not follow "well-behaved" trend seen previously (Figure 12a). Instead, the energy increased to almost initial energy before eventually

settling into equilibrium. System geometries were inspected to find problems with the MD run. After 1.5 ps into MD simulation, water molecules evaporated because the size of the solvent shell was not large enough to form a fully Hydrogen-bonded network around the solute. During the evaporation, the potential energy of the system increased momentarily, before slowly decreased as the system equilibrated. This emphasized a problem in running simulation at room temperature which provided enough potential energy for water molecules to evaporate from the water sphere. A method of using a potential boundary was proposed to

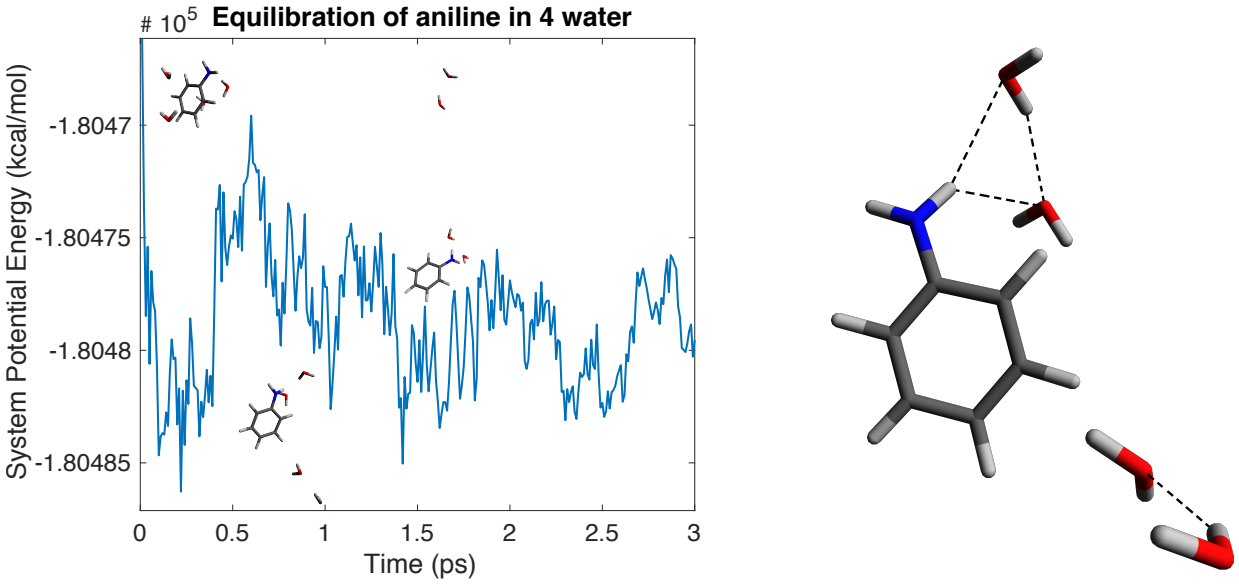


Figure 15: Molecular dynamics of aniline with 4 water without solvent boundary condition. a) A plot of system energy as a function of time with system geometries at 0, 0.43, and 1.5 picoseconds. Notice the by 0.4 ps, the two water already started evaporating out from solute b) Hydrogen bonding between hydrogen and oxygen atoms in aniline with 4 water 0.4 ps into the MD simulation. The H-bonding ranged from 2.2 to 2.8 Angstrom

simulate the bulk solvent force field around the solute.[85] When a solvent fragment crosses a empirically-assigned radius, a potential "wall" will exert slight force to push the molecule back toward the solvent sphere. The Spherical Solvent Boundary Potential (SSBP) radius is set empirically from trial and error (see Appendix for prepareMD2.py for estimating SSBP radius). In the ab initio quantum chemistry package GAMESS[72], the potential is static and does not move as the solute molecule translates. The potential boundary should not influence the structure of the system, but rather should serve as a preventive measure against evaporation. Solvent boundary condition is implemented, albeit incompletely, in GAMESS with command SSBP=.T (see Appendix). The function for boundary potential is

$$V = 0.5 \times SFORCE \times (r_{SSBP})^2 \tag{17}$$

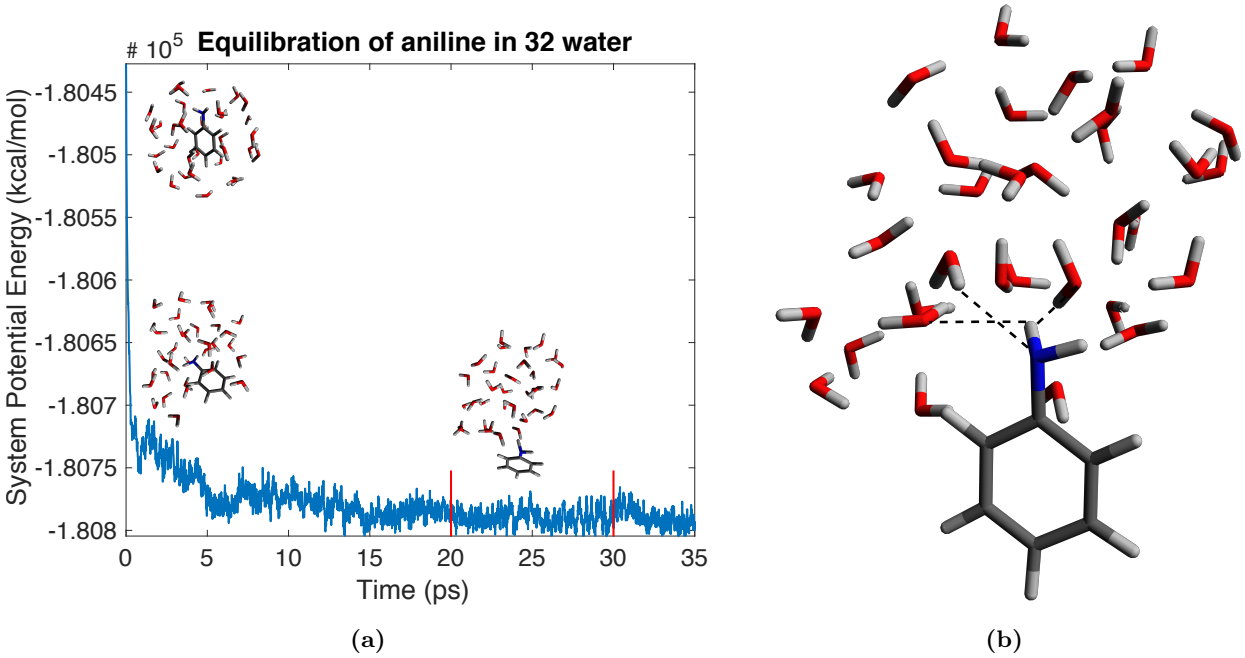


Figure 16: Molecular dynamics of aniline with 32 water without solvent boundary condition. a) A plot of system energy as a function of time with system geometries at 0, 2.5, and 20 ps. Notice the by 2.5 ps, the solute already started to get move out of the sphere b) Hydrogen bonding between hydrogen and oxygen atoms in aniline at 30 ps into the MD simulation. This showed that even though aniline was not stable being surrounded by 32 water molecules, the molecule does not completely lose contact with the solvent. More water molecules would indeed more fully solvate aniline.

Alternatively, the possibility of implementing periodic boundary conditions[86] should also be explored in the future. Periodic boundary conditions model an object on the other side of the box as the object moves through a wall. By limiting the size of solvent box, periodic boundary condition may allow for a more logical model compared with solvent boundary potentials. Both of these techniques are implemented in GAMESS.

4.1.2 Aniline With 32 Water Molecules

32-water model is the smallest solvent model in this study. When used with aniline, MD system energy is also not "well-behaved" (see Figure 16a). At 1ps and 7 ps, system energy increased slightly before decreased back down again. Even though these increases was not to the same degree as in 4-water case, a closer look at system geometry revealed a problem with this MD calculation. Aniline molecule appeared to be moving out of the water sphere at 2.5 ps. At equilibrium, aniline molecule could be seen pushed outside of the water cluster. This fact contradict the conclusion from Plugatyr that 32 water molecules should represent the first solvation shell for aniline.[87] If aniline molecule was not stable being surrounded by 32 water molecules, then the number of water molcules in the first solvation shell should be higher than 32. Thus, MD-TD-DFT of results from 32-water model be analyzed with caution due to incomplete solvation shell.

4.2 Determining The Optimum Solvent Environment

Experimental results and MD-TD-DFT excited states energy and oscillator strength pairs for aniline with 32, 64, 128, 256, and 512 surrounding water molecules were tabulated in Table 3. Clearer representation of the data is an area plot in Figure 17 with excitation energy (E, nm) on the x axis, and the oscillator strength (f, no unit) as an area of the data point.

Table 3: Wavelength and oscillator strength from MD-TD-DFT calculation of aniline in 32-512 water molecules. Energy has unit of nm. Oscillator strength has no unit.

Experimental		**32-water**		**64-water**		**128-water**		**256-water**		**512-water**	
E	f	E	f	E	f	E	f	E	f	E	f
204.5	0.1578	193.6	0.0842	167.9	0.0293	161.0	0.0668	160.9	0.0378	161.1	0.0627
223.6	0.1067	205.3	0.0079	182.3	0.3391	180.2	0.4280	180.5	0.3984	181.9	0.4046
233.3	0.1049	221.3	0.0975	203.7	0.0573	212.5	0.1226	217.1	0.1730	216.3	0.1437
280.0	0.0269	224.4	0.0157	233.1	0.0627	241.7	0.0343	250.1	0.0446	240.9	0.0666
		227.0	0.0195							247.4	0.0400
		229.3	0.0829								
		230.3	0.0392								
		253.2	0.0361								

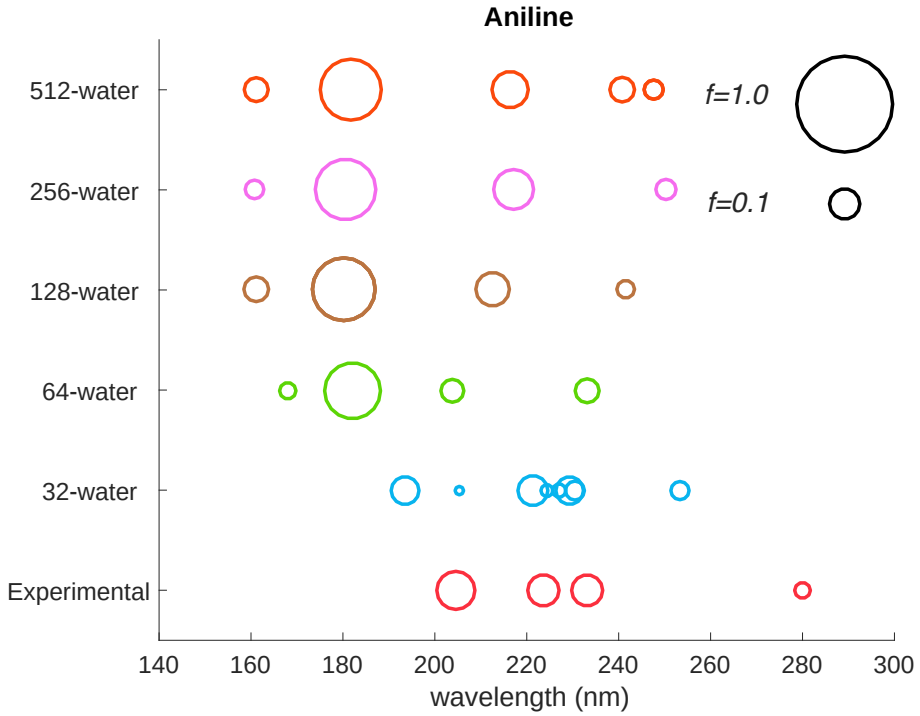


Figure 17: Predicted excitation energies (x-axis) and oscillator strength (area of marker) varying by the number of explicit water molecules using CAM-B3LYP functional. The two black dots on the top right shows the size of circle representing oscillator strength = 0.1 and 1. Different colors represent results from different water models.

Energy and oscillator strength values calculated from the 64-water model to 512-water model showed

convergence toward the values from 256-water model. (E, f) pairs of the absorption near 210 nm for 64 to 512-model were (203.7 nm, 0.0573), (212.5 nm, 0.1226), (217.1 nm, 0.1730), (216.3 nm, 0.1437). 256-model being the most redshifted and the most probable transition. As the number of solvent molecules increases, excited state energy from the models converged to 217 nm. Convergence in other electronic transitions in the circle plot could also be observed. Theoretical results started to converge beginning from 64-water to the values in 256 and 512-water model. 128, 256, and 512-water models gave similar results with 256-water being the most redshifted. Note that the two data points with similar values of E and f could be the same mode of excitation separated by faulty statistical analysis. For example, while other models only had one electronic transition at 240 nm, 512-model yielded two at (240.9 nm, 0.0666) and (247.4 nm, 0.0400). Matlab might put two Gaussian fits in 512-model but one in other models according to Bayesian statistics. This potential problem was acknowledged and will be explored in the next versions of Matlab code. The lowest energy transition in the 128-water model was still converging to the value of 256 and 512. It was noted that compared among theoretical data, 32-water model failed to correlate with the trend observed in models with more water molecules. 32-water model's downfall could be explained by a quick inspection at the system equilibrated geometries as discussed above. When compared with results from the experiment, theoretical results from CAM-B3LYP with any number of water molecules were mostly inaccurate. Experimental data ranged from 200 nm to 300 nm. All models missed to predict one particular transition of aniline at 280 nm. This could be because of CAM-B3LYP's failure at predicting a low-energy transition state. This problem will be discussed in the next Section.

More results in addition to the (E, f) calculated from the equilibrated region needed to be examined to determine the whether 128, 256, or 512 water molecules best model solute-solvent effects without too much cost. More excited state energies and oscillator strengths were calculated using a range of 128 to 512-model geometries obtained from MD. The results are then grouped by 10 ps time interval starting at 0 ps. This would show how quick the excited state energies and oscillator strengths value converge to the values calculated from geometries at the equilibrium. The values were calculated from geometries with interval of 0.5 ps in range of 10 ps. Tabulated data can be found in Table 4 in the Appendix. In Figure 18b, starting form 20-30 ps, the two transitions at 220 and 250 nm disappeared. This was speculated to be from Matlab's statistical method. When two transitions are close, depending on the sample, Matlab might fit one or two Gaussian components in. Excited state energy and oscillator strength values seemed to converge long before the currently determined equilibrium. In Figure 18a, the results from 128-water-model aniline were stable through out the time span and did not seem to converge as time progressed. This indicated that the equilibrium was established very early. On the other hand, 256-water-model aniline results did show

convergence in low-energy transitions. However, the apparent convergence could be a defect from Bayesian statistical analysis. In contrast, results from 512-model converged more slowly compared to 128 and 256-model. Note the fluctuation in data from 0-10 ps to 40-50 ps. Unlike, the previous two cases, 512-model had more inconsistency. Furthermore, MD calculation of 512-model took significantly longer time compared to 256-model. Combining this to the results from Figure 17, 256-water model was chosen to apply to other molecules.

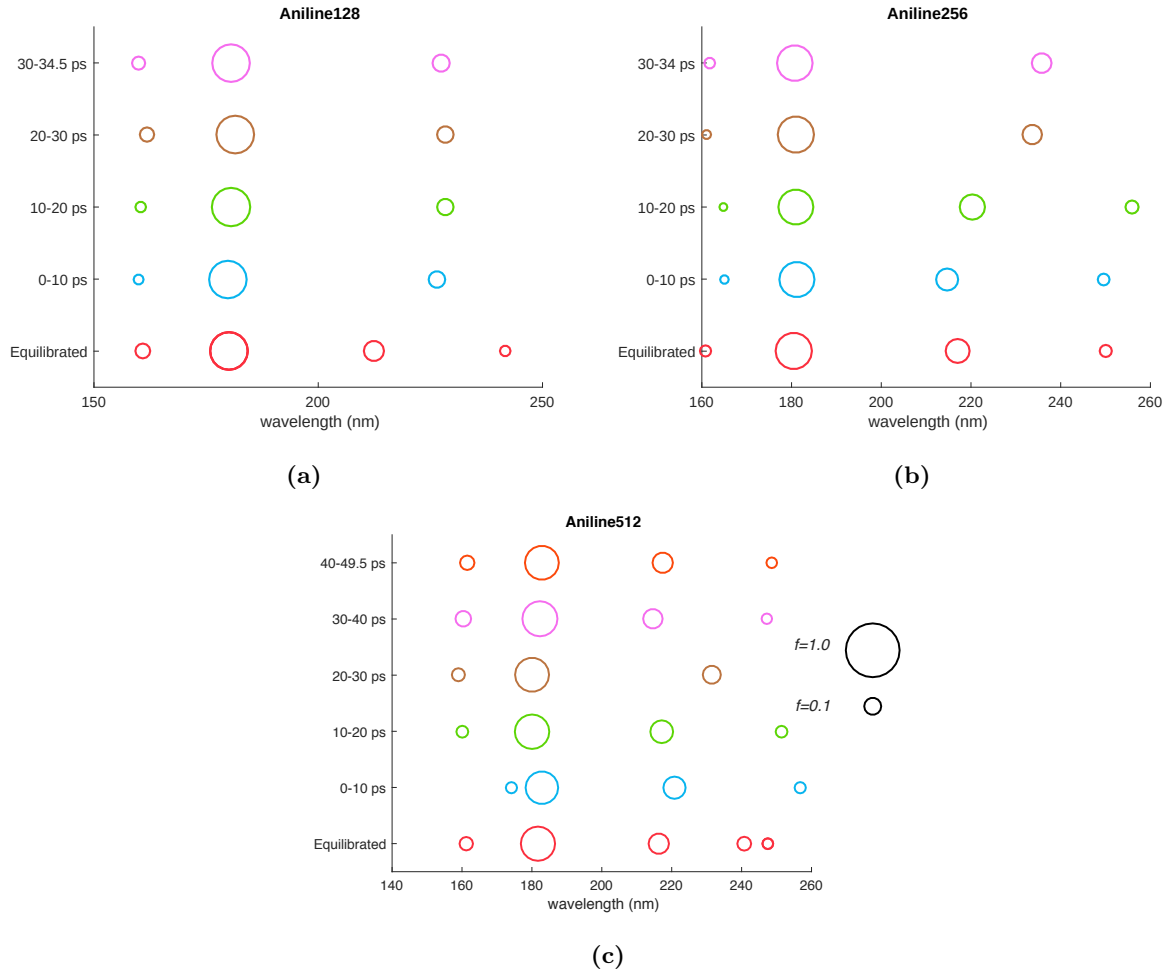


Figure 18: Predicted excitation energies and oscillator strengths varying by time (y-axis) for aniline in 128, 256, 512-water model

4.3 Comparing Different Functionals

MD-TD-DFT excited states of aniline, 3-F-aniline, 4-F- aniline, 3-Cl-aniline, 4-MeOacetophenone, and (1,3)-dimethoxybenzophenone using LCBLYP, BP86, M06-2X, PBE0, and CAM-B3LYP functionals were calculated and compared with the experimental results. In the environment, the main source of photon is the

sun. The absorption spectrum overlapping with the solar irradiance plot showed that low-energy electronic transitions above 250 nm are more relevant to modeling the light absorption of pollutants in aquatic environment (Figure 5). The circle plots in Figure 19 then had a domain starting from 250 nm to focus more on important electronic transition in nature. Table of the same data can be found in 5 in the Appendix. Most surprisingly, all functionals except BP86 missed low-energy transitions. Despite some small errors, BP86 was the best-performing functional. In Figure 19, BP86 in green calculated low energy transitions that other functionals failed to calculate. For aniline, for example, CAM-B3LYP, PBE0, and LCBLYP all predicted their lowest-energy transition at 250 nm, in contrast to BP86 and experimental data which predicted a transition at 280 nm. BP86's lowest-energy excited state had the error in energy of 3.5 nm and error in oscillator strength of 0.005. In 3-F-aniline case, there was no other functional other than BP86 that predicted low energy transition at 275 nm with the error of 4 nm and 0.007. For 4-MeOacetophenone, BP86's the second lowest excited state had the error of 0.5 nm and 0.03. BP86 is unique in this group for being the only pure GGA-type in all functionals considered. There is no other specific reason why BP86 might perform better than other functionals. BP86 s data in full range area plot, however, showed a consistent redshift in excited state energy calculation relative to other functionals. There were two possible explanation to this. It could mean BP86 performed badly compared to other functionals chosen here because it underestimated the excited energies and that all MD-TD-DFT excited states are blueshifted significantly. It could also mean all other hybrid functionals are inaccurate compared to pure GGA BP86 and the experimental results. If the former, improvement on current model such as EFP2[88] and dispersion correction[89] can reduce blueshift error by more correctly treat solvent-solute interaction. If the latter, more pure GGA functionals such as BPBE[60,90] and BLYP[60,63] should be explored.

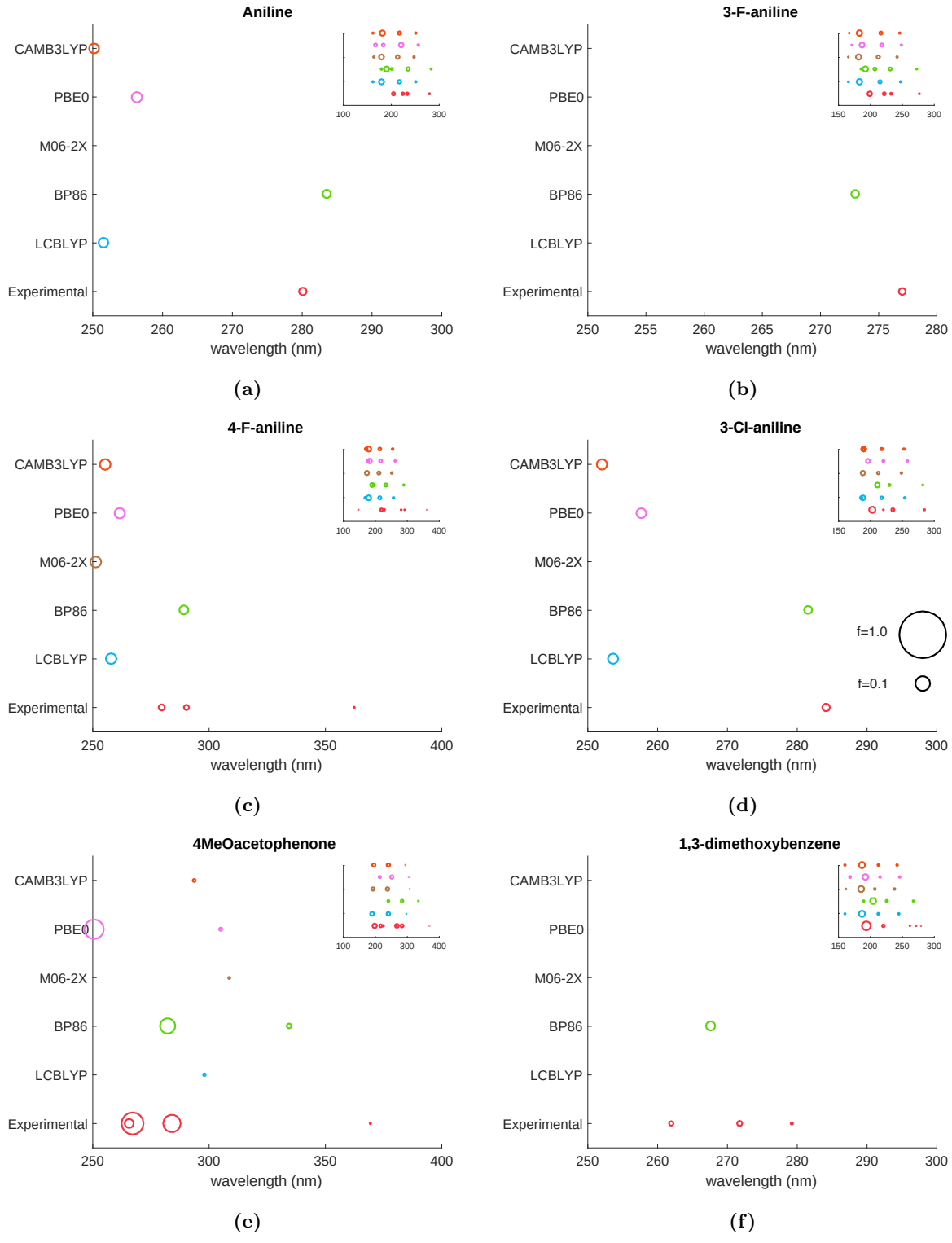


Figure 19: Predicted excitation state energies and oscillator strength varying by DFT functionals starting from 250 nm with full Figure in a small box on the top right. A full Figure and full data can be seen in appendix.

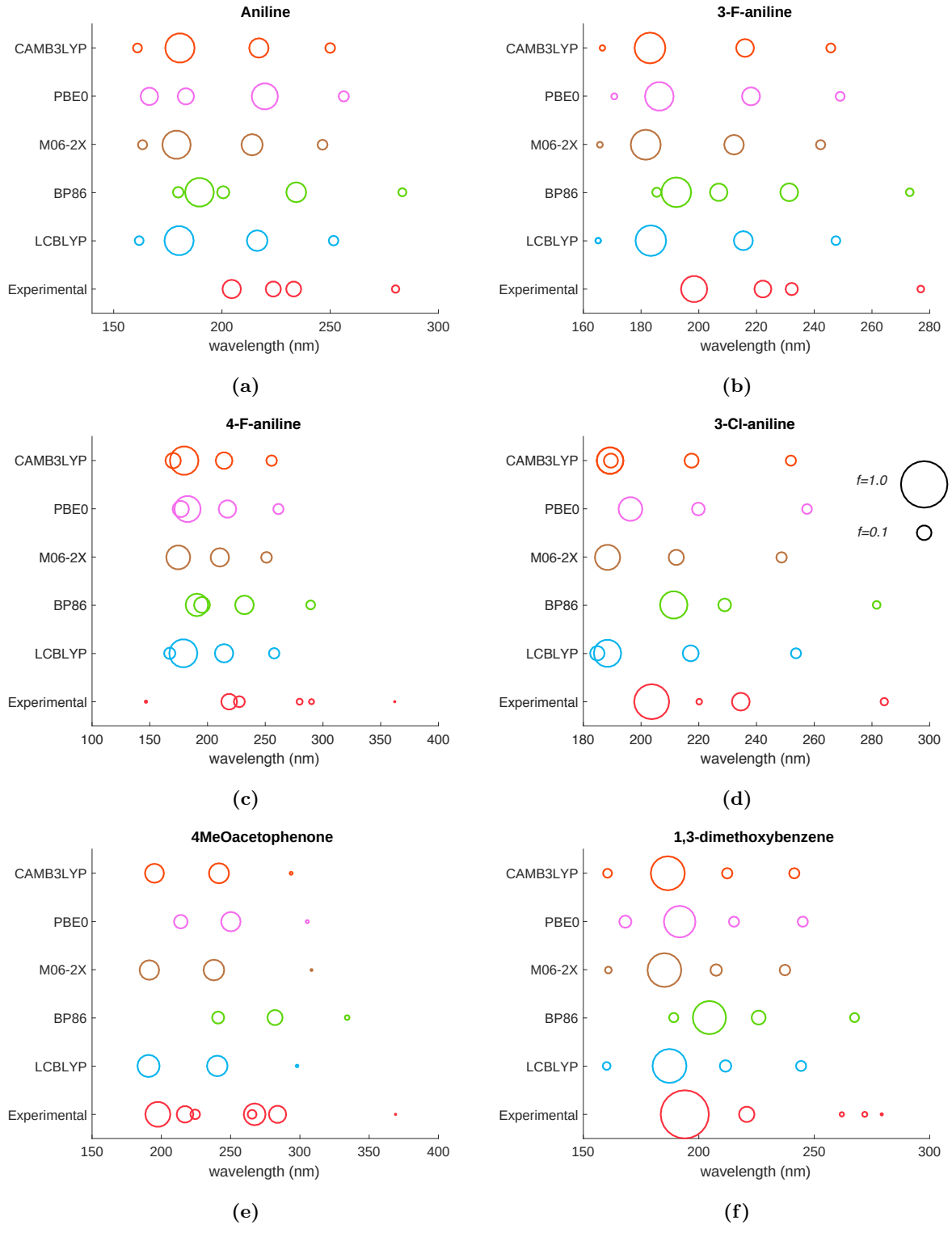


Figure 20: Predicted excitation state energies and oscillator strength varying by DFT functionals.

5 Conclusion

An ideal model for calculating UV-VIS absorption spectrum has not yet been found, but a protocol and method of study to find it was proposed (see flowchart in Figure 10). In pioneering a project, development of method of work and system is crucial to the future study in the lab. For the years to come, Python scripts, Matlab codes and protocol will be used to continue searching for an ideal model to model photo-absorption part of the sunlight-dirven pollutant degradation in aquatic environments. For the first time, a systematic statistical approach was used to compare experimental molar absorptivity with theoretical MD-TD-DFT oscillator strength. Despite some issues with Matlab Bayesian criteria, a possibility has been demonstrated that a sensible set of data can be calculated. An approach to find the optimum number of explicit solvent molecules was also outlined. For small molecules like aniline, it was shown that 256 water molecules serve as a good trade-off between computational cost and accuracy. MD equilibrium determination was still a problem in the current protocol. MD-TD-DFT results demonstrated that the proposed method of equilibrium determination using findEquilibrium.py with 10 picoseconds as equilibration period is incorrect. There is then a need for statistical method to determine the equilibrium.

Even though no DFT functional was shown to have give results matching exactly with the experimental results, BP86 was identified as the best-performing functional for MD-TD-DFT calculation on aniline, 3-F-aniline, 4-F- aniline, 3-Cl-aniline, 4-MeOacetophenone, and (1,3)-dimethoxybenzophenone. More pure GGA and hybrid functionals should be explored and benchmarked with experimental results to observe more trends with different group of functionals. Larger molecules perhaps some well-studied micropollutants should be used for MD-TD-DFT calculation using the protocol proposed.

Proceeding forward, these initial results can be used to further move closer to the overall goal of developing fully computational model to predict behavior of pollutant in aquatic environment. When excited states energy and probability of excitation are accurately predicted, rate constant of photon absorption can be calculated, and combined with other data, used to predict the rate of photo-reactions.

Acknowledgment

I would like to express my deepest appreciation professor Soren Eustis for being such a best mentor and helping me on almost all aspects of working within and without the lab. I wish to acknowledge the contribution provided by Alex Poblete, Holly Rudel, and Peter Cohen who have worked in Eustis lab and provide experimental data and mathematical insight to the project. I would like to offer my special thanks to Dj Merrill for supervising Bowdoin HPC and helping with troubleshooting throughout the years. I would like to thank James Stacy Coles / Littlefield Summer Research Fellowships, Grua/OConnell Fund, XSEDE project funded by NSF award number ACI 10-53575 for all the funding that comes to this project.

References

(1) Schwarzenbach, R. . P.; Escher, B. I.; Fenner, K.; Hofstetter, T. B.; Johnson, C. A.; von Gunten, U.; Wehrli, B. The Challenge of Micropollutants in Aquatic Systems. *Science* **2006**, *313*, 1072–1077.

(2) Monteiro, S.; Boxall, A. *Reviews of Environmental Contamination and Toxicology SE - 2*; Reviews of Environmental Contamination and Toxicology; Springer New York, 2010; Vol. 202; pp 53–154.

(3) Harada, M. Minamata disease: methylmercury poisoning in Japan caused by environmental pollution. *Critical reviews in toxicology* **1995**, *25*, 1–24.

(4) Original image copyright belongs to noodledoodle and zhaolifang from vecteezy.com: non-exclusive right to use Eezy files to create new, derivative, unique works.

(5) Conley, J. M.; Symes, S. J.; Schorr, M. S.; Richards, S. M. Spatial and temporal analysis of pharmaceutical concentrations in the upper Tennessee River basin. *Chemosphere* **2008**, *73*, 1178–1187.

(6) Daneshvar, A.; Svanfelt, J.; Kronberg, L.; Weyhenmeyer, G. A. Winter accumulation of acidic pharmaceuticals in a Swedish river. *Environmental Science and Pollution Research* **2010**, *17*, 908–916.

(7) Bonvin, F.; Rutler, R.; Chavre, N.; Halder, J.; Kohn, T. Spatial and temporal presence of a wastewater-derived micropollutant plume in Lake Geneva. *Environmental Science and Technology* **2011**, *45*, 4702–4709.

(8) Carlson, J. C.; Stefan, M. I.; Parnis, J. M.; Metcalfe, C. D. Direct UV photolysis of selected pharmaceuticals, personal care products and endocrine disruptors in aqueous solution. *Water Research* **2015**, *84*, 350–361.

(9) Kliegman, S.; Eustis, S. N.; Arnold, W. a.; McNeill, K. Experimental and theoretical insights into the involvement of radicals in triclosan phototransformation. *Environmental science & technology* **2013**, *47*, 6756–63.

(10) Bedoux, G.; Roig, B.; Thomas, O.; Dupont, V.; Le Bot, B. Occurrence and toxicity of antimicrobial triclosan and by-products in the environment. *Environmental Science and Pollution Research* **2012**, *19*, 1044–1065.

(11) Morss, P. C. The Photodegradation of Endocrine Disrupting Compounds in Aqueous Solutions. Ph.D. thesis, Bowdoin College, 2014.

(12) Salvato-Vallverdu, G. The Perrin - Jablonski diagram. 2009.

(13) Hirayama, S.; Steer, R. P. Understanding the Theory and Practice of Molecular Spectroscopy: The Effects of Spectral Bandwith. *Journal of Chemical Education* **2010**, *87*, 1344–1347.

(14) Reichardt, C.; Welton, T. *Solvents and Solvent Effects in Organic Chemistry*; Wiley, 2011.

(15) Turro, N. J.; Ramamurthy, V.; Scaiano, J. C. *Modern Molecular Photochemistry of Organic Molecules*; University Science Books, 2010.

(16) Clemens, O.; Basters, M.; Wild, M.; Wilbrand, S.; Reichert, C.; Bauer, M.; Springborg, M.; Jung, G. Solvent effects on the absorption/emission spectra of an organic chromophore: A theoretical study. *Journal of Molecular Structure: THEOCHEM* **2008**, *866*, 15–20.

(17) Suendo, V.; Viridi, S. Ab initio calculation of UV-Vis absorption spectra of a single chlorophyll a molecule: comparison study between RHF/CIS, TDDFT, and semi-empirical methods. *ITB Journal of Science* **2012**, *44A*, 93–112.

(18) Kakitani, T.; Mataga, N. Comprehensive study on the role of coordinated solvent mode played in electron-transfer reactions in polar solutions. *The Journal of Physical Chemistry* **1987**, *91*, 6277–6285.

(19) Marcus, R. A. Relation between Charge Transfer Absorption and Fluorescence Spectra and the Inverted Region. *The Journal of Physical Chemistry* **1989**, *93*, 3078–3086.

(20) Matyushov, D. V. Solvent reorganization energy of electron-transfer reactions in polar solvents. *Journal of Chemical Physics* **2004**, *120*, 7532–7556.

(21) Barthel, E. R.; Martini, I. B.; Schwartz, B. J. How does the solvent control electron transfer? Experimental and theoretical studies of the simplest charge transfer reaction. *Journal of Physical Chemistry B* **2001**, *105*, 12230–12241.

(22) Eilmes, A. Solvatochromic probe in molecular solvents: implicit versus explicit solvent model. *Theoretical Chemistry Accounts* **2014**, *133*, 1–13.

(23) Ricke, N. D. Application of the Landau-Zener Model and Fermi's Golden Rule to Estimate Triplet Quantum Yield for Organic Molecules. Ph.D. thesis, Bowdoin College, 2014.

(24) Leifer, A. *The Kinetics of Environmental Aquatic Photochemistry: Theory and Practice*; ACS professional reference book; American Chemical Society, 1988.

(25) Schwarzenbach, R. P.; Gschwend, P. M.; Imboden, D. M. *Environmental Organic Chemistry*; Wiley, 2005.

(26) ASTM G173-03(2012), Standard Tables for Reference Solar Spectral Irradiances: Direct Normal and Hemispherical on 37 Tilted Surface, ASTM International, West Conshohocken, PA, 2012, www.astm.org.

(27) Dykstra, C. E.; Frenking, G.; Kim, K. S.; Scuseria, G. E. *Theory and Applications of Computational Chemistry: The First Forty Years*; Elsevier, 2011.

(28) Bartlett, R. J.; Purvis, G. D. Many-body perturbation theory, coupled-pair many-electron theory, and the importance of quadruple excitations for the correlation problem. *International Journal of Quantum Chemistry* **1978**, *14*, 561–581.

(29) Pople, J.; Krishnan, R.; Schlegel, H. B.; Binkley, J. S. Electron correlation theories and their application to the study of simple reaction potential surfaces. *International Journal of Quantum Chemistry* **1978**, *XIV*, 545.

(30) Frisch, M. J., et al. (2009). Gaussian 09. Wallingford, CT, USA, Gaussian, Inc.

(31) Eisberg, R.; Resnick, R. *Quantum physics of atoms, molecules, solids, nuclei, and particles*, 2nd ed.; John Wiley & Sons: New York, 1985.

(32) Griffiths, D. J. *Introduction to Quantum Mechanics*; Pearson Education India, 2005.

(33) Møller, C.; Plesset, M. S. Note on an approximation treatment for many-electron systems. *Physical Review* **1934**, *46*, 618–622.

(34) Hehre, W. J.; Radom, L.; Schleyer, P. v. R.; Pople, J. *AB INITIO Molecular Orbital Theory*; A Wiley-Interscience publication; Wiley, 1986.

(35) Magalhães, A. L. Gaussian-type orbitals versus slater-type orbitals: A comparison. *Journal of Chemical Education* **2014**, *91*, 2124–2127.

(36) Hehre, W. J.; Stewart, R. F.; Pople, J. A. SelfConsistent MolecularOrbital Methods. I. Use of Gaussian Expansions of SlaterType Atomic Orbitals. *The Journal of Chemical Physics* **1970**, *52*, 2769.

(37) Cramer, C. J. *Essentials of Computational Chemistry: Theories and Models*, 2nd ed.; Wiley, 2005.

(38) Rohrdanz, M. A.; Herbert, J. M. Simultaneous benchmarking of ground- and excited-state properties with long-range-corrected density functional theory. *Journal of Chemical Physics* **2008**, *129*, 1–9.

(39) Barnes, L.; Abdul-Al, S.; Allouche, A.-R. TDDFT Assessment of Functionals for Optical 00 Transitions in Small Radicals. *The Journal of Physical Chemistry A* **2014**, *118*, 11033–11046.

(40) Wiberg, K. B. Basis set effects on calculated geometries: 6-311++G** vs. aug-cc-pVDZ. *Journal of Computational Chemistry* **2004**, *25*.

(41) Feller, D. The Role of Databases in Support of Computational Chemistry Calculations. *Journal of Computational Chemistry* **1996**, *17*, 1571–1586.

(42) Baerends, E. J.; Gritsenko, O. V. A quantum chemical view of density functional theory. *Journal of Physical Chemistry A* **1997**, *101*, 5383–5403.

(43) Sousa, S. F.; Fernandes, P. A.; Ramos, M. J. General performance of density functionals. *Journal of Physical Chemistry A* **2007**, *111*, 10439–10452.

(44) Hohenberg, P.; Kohn, W. Inhomogeneous Electron Gas. *Physical Review* **1964**, *136*, B864–B871.

(45) Kohn, W.; Sham, L. J. Self-Consistent Equations Including Exchange and Correlation Effects. *Physical Review* **1965**, *140*, A1133–A1138.

(46) Heßelmann, A. Molecular excitation energies from time-dependent density functional theory employing random-phase approximation Hessians with exact exchange. *Journal of Chemical Theory and Computation* **2015**, *11*, 1607–1620.

(47) Hirao, H. Which DFT functional performs well in the calculation of methylcobalamin? Comparison of the B3LYP and BP86 functionals and evaluation of the impact of empirical dispersion correction. *Journal of Physical Chemistry A* **2011**, *115*, 9308–9313.

(48) Parac, M.; Grimme, S. Comparison of multireference Møller-Plesset theory and time-dependent methods for the calculation of vertical excitation energies of molecules. *Journal of Physical Chemistry A* **2002**, *106*, 6844–6850.

(49) Jacquemin, D.; Wathelet, V.; Perpète, E. a.; Adamo, C. Extensive TD-DFT benchmark: Singlet-excited states of organic molecules. *Journal of Chemical Theory and Computation* **2009**, *5*, 2420–2435.

(50) Caricato, M.; Trucks, G. W.; Frisch, M. J.; Wiberg, K. B. Electronic transition energies: A study of the performance of a large range of single reference density functional and wave function methods on valence and rydberg states compared to experiment. *Journal of Chemical Theory and Computation* **2010**, *6*, 370–383.

(51) Bersier, C.; Proetto, C.; Sanna, A.; Cangi, A.; Mirhosseini, S. H. Basic rules for functional construction at finite temperature. Ph.D. thesis, Max Planck Institute of Microstructure Physics.

(52) Yanai, T.; Tew, D. P.; Handy, N. C. A new hybrid exchange correlation functional using the Coulomb-attenuating method (CAM-B3LYP). *Chemical Physics Letters* **2004**, *393*, 51–57.

(53) Zhao, Y.; Truhlar, D. G. The M06 suite of density functionals for main group thermochemistry , thermochemical kinetics , noncovalent interactions , excited states , and transition elements : two new functionals and systematic testing of four M06-class functionals and 12 other fun. *Theoretical Chemistry Accounts* **2007**, *120*, 215–241.

(54) Adamo, C.; Barone, V. Toward reliable density functional methods without adjustable parameters: The PBE0 model. *The Journal of Chemical Physics* **1999**, *110*, 6158.

(55) Magyar, R. J.; Tretiak, S. Dependence of spurious charge-transfer excited states on orbital exchange in TDDFT: Large molecules and clusters. *Journal of Chemical Theory and Computation* **2007**, *3*, 976–987.

(56) Runge, E.; Gross, E. K. U. Density-functional theory for time-dependent systems. *Physical Review Letters* **1984**, *52*, 997–1000.

(57) Caricato, M.; Mennucci, B.; Tomasi, J. Solvent Effects on the Electronic Spectra: An Extension of the Polarizable Continuum Model to the ZINDO Method. *The Journal of Physical Chemistry A* **2004**, *108*, 6248–6256.

(58) Fabian, J. Electronic excitation of sulfur-organic compounds - Performance of time-dependent density functional theory. *Theoretical Chemistry Accounts* **2001**, *106*, 199–217.

(59) Barone, V.; Polimeno, A. Integrated computational strategies for UV/vis spectra of large molecules in solution. *Chemical Society Reviews* **2007**, *36*, 1724.

(60) Becke, A. D. Density-functional exchange-energy approximation with correct asymptotic behavior. *Physical Review A* **1988**, *38*, 3098–3100.

(61) Perdew, J. P. Density-functional approximation for the correlation energy of the inhomogeneous electron gas. *Physical Review B* **1986**, *33*, 8822–8824.

(62) Iikura, H.; Tsuneda, T.; Yanai, T.; Hirao, K. A long-range correction scheme for generalized-gradient-approximation exchange functionals. *Journal of Chemical Physics* **2001**, *115*, 3540–3544.

(63) Lee, C.; Yang, W.; Parr, R. G. Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density. *Physical Review B* **1988**, *37*, 785–789.

(64) Lin, H.; Truhlar, D. G. QM/MM: what have we learned, where are we, and where do we go from here? *Theoretical Chemistry Accounts* **2007**, *117*, 185–199.

(65) Cossi, M.; Barone, V. Solvent effect on vertical electronic transitions by the polarizable continuum model. *The Journal of Chemical Physics* **2000**, *112*, 2427.

(66) Mennucci, B. Continuum solvation models: What else can we learn from them? *Journal of Physical Chemistry Letters* **2010**, *1*, 1666–1674.

(67) Tomasi, J.; Mennucci, B.; Cammi, R. Quantum mechanical continuum solvation models. *Chemical Reviews* **2005**, *105*, 2999–3093.

(68) Li, J.; Cramer, C. J.; Truhlar, D. G. Two-Response-Time Model Based on CM2/INDO/S2 Electrostatic Potentials for the Dielectric Polarization Component of Solvatochromic Shifts on Vertical Excitation Energies. *International Journal of Quantum Chemistry* **1999**, *77*, 264–280.

(69) Day, P. N.; Jensen, J. H.; Gordon, M. S.; Webb, S. P.; Stevens, W. J.; Krauss, M.; Garmer, D.; Basch, H.; Cohen, D. An effective fragment method for modeling solvent effects in quantum mechanical calculations. *The Journal of Chemical Physics* **1996**, *105*, 1968–1986.

(70) Yoo, S.; Zahariev, F.; Sok, S.; Gordon, M. S. Solvent effects on optical properties of molecules: A combined time-dependent density functional theory/effective fragment potential approach. *Journal of Chemical Physics* **2008**, *129*, 1–8.

(71) Defusco, A.; Minezawa, N.; Slipchenko, L. V.; Zahariev, F.; Gordon, M. S. Modeling Solvent Effects on Electronic Excited States. *Journal of Physical Chemistry Letters* **2011**, *2*, 2184–2192.

(72) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Shyjun, S. U. et al. General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry* **1993**, *14*, 1347–1363.

(73) Chodera, J. D. A simple method for automated equilibration detection in molecular simulations. *Journal of Chemical Theory and Computation* **2015**, 021659.

(74) Hilborn, R. C. Einstein coefficients, cross sections, f values, dipole moments, and all that. *American Journal of Physics* **1982**, *50*, 982–986.

(75) Klán, P.; Wirz, J. *Photochemistry of organic compounds: From concepts to practice*; John Wiley & Sons, 2009.

(76) Python Software Foundation. Python Language Reference, version 2.7.

(77) Martínez, L.; Andrade, R.; Birgin, E. G.; Martínez, J. M. Software News and Update Packmol : A Package for Building Initial Configurations. *Journal of Computational Chemistry* **2009**, *30*, 2157–2164.

(78) Matlab version 7.14.0.739. Natick, Massachusetts, The MathWorks Inc.

(79) Schmidt, J.R.; Polik, W.F. WebMO Enterprise, version 13.0; WebMO LLC: Holland, MI, USA, 2013; http://www.webmo.net (accessed March, 2013).

(80) Wojdyr, M. Fityk: A general-purpose peak fitting program. *Journal of Applied Crystallography* **2010**, *43*, 1126–1128.

(81) Eustis, S. N.; Trerayapiwat, K.; Ricke, N.; Cohen, P.; Poblete, A.; Rudel, H. Computational Environmental Photochemistry: A Roadmap for Applied Quantitative Calculations. *Royal Chemical Society* **2016**,

(82) Armarego, W. L. F.; Chai, C. L. L. *Purification of laboratory chemicals*, 7th ed.; Butterworth-Heinemann, 2013.

(83) Yalkowsky, S. H.; He, Y.; Jain, P. *Handbook of Aqueous Solubility Data, Second Edition*; CRC Press, 2016.

(84) R Development Core Team, R: A Language and Environment for Statistical Computing. 2008.

(85) Beglov, D.; Roux, B. Finite representation of an infinite bulk system: Solvent boundary potential for computer simulations. *The Journal of Chemical Physics* **1994**, *100*.

(86) Makov, G.; Payne, M. Periodic boundary conditions in ab initio calculations. *Physical Review B* **1995**, *51*, 4014–4022.

(87) Plugatyr, A.; Svishchev, I. M. The hydration of aniline: Analysis of spatial distribution functions. *The Journal of chemical physics* **2009**, *130*, 114509.

(88) Gordon, M. S.; Freitag, M. a.; Bandyopadhyay, P.; Jensen, J. H.; Kairys, V.; Stevens, W. J. The Effective Fragment Potential Method : A QM-Based MM Approach to Modeling. *Journal of Physical Chemistry A* **2001**, *105*, 293–307.

(89) Guidez, E. B.; Gordon, M. S. Dispersion Correction Derived from First Principles for Density Functional Theory and HartreeFock Theory. *The Journal of chemical physics* **2015**, *119*, 21612168.

(90) Perdew, J. P.; Burke, K.; Ernzerhof, M. Generalized Gradient Approximation Made Simple. *Physical Review Letters* **1996**, *77*, 3865–3868.

A Appendix

A.1 MD-TD-DFT Result Tables

Table 4: Wavelength and Oscillator Strength from MD-TD-DFT calculation of aniline in 128-512 water molecules as a function of time.

128-water

Experimental		0-10 ps		10-20 ps		20-30 ps		30-34.5 ps	
E	f	E	f	E	f	E	f	E	f
160.98	0.0668	159.88	0.0302	160.40	0.0345	161.93	0.0624	160.03	0.0522
180.20	0.4280	179.77	0.4255	180.53	0.4452	181.42	0.4297	180.57	0.4269
212.48	0.1226	226.39	0.0827	228.26	0.0815	228.35	0.0827	227.48	0.0894
241.70	0.0343								

256-water

Experimental		0-10 ps		10-20 ps		20-30 ps		30-34 ps	
E	f	E	f	E	f	E	f	E	f
160.90	0.0378	165.05	0.0224	164.81	0.0190	161.04	0.0246	161.83	0.0353
180.51	0.3984	181.14	0.3726	180.85	0.3690	180.94	0.3967	180.67	0.3826
217.07	0.1730	214.77	0.1465	220.35	0.1925	233.71	0.1150	235.77	0.1177
250.13	0.0446	249.57	0.0421	255.88	0.0522				

512-water

Experimental		0-10 ps		10-20 ps		20-30 ps		30-40 ps		40-49.5 ps	
E	f	E	f	E	f	E	f	E	f	E	f
161.09	0.0627	174.24	0.0435	160.16	0.0489	158.87	0.0592	160.27	0.0847	161.46	0.0724
181.86	0.4046	182.99	0.3670	179.95	0.4100	179.93	0.3994	182.21	0.4254	182.84	0.3975
216.35	0.1437	220.86	0.1708	217.26	0.1790	231.48	0.1126	214.56	0.1313	217.49	0.1437
240.86	0.0666	256.85	0.0438	251.30	0.0481			247.13	0.0380	248.74	0.0394
247.44	0.0400										

Table 5: Wavelength and Oscillator Strength from MD-TD-DFT calculation of aniline, 3-F-aniline, 4-F-aniline, 3-Cl-aniline, 4-MeOacetophenone, and (1,3)-dimethoxybenzophenone using LCBLYP, BP86, M06-2X, PBE0, and CAM-B3LYP functionals.

Experiment		**LCBLYP**		**BP86**		**M06-2X**		**PBE0**		**CAM-B3LYP**	
Aniline											
E	f	E	f	E	f	E	f	E	f	E	f
204.45	0.1578	161.76	0.0370	179.76	0.0531	163.29	0.0398	166.29	0.0523	160.90	0.0378
223.58	0.1067	180.22	0.3946	189.71	0.3844	179.08	0.3725	183.53	0.3305	180.51	0.3984
233.29	0.1049	216.38	0.1974	200.74	0.0694	213.88	0.2086	219.99	0.1818	217.07	0.1730
280.05	0.0269	251.51	0.0433	234.25	0.1829	246.59	0.0461	256.27	0.0419	250.13	0.0446
				283.49	0.0314						
3-F-aniline											
E	f	E	f	E	f	E	f	E	f	E	f
198.42	0.3185	165.18	0.0144	185.47	0.0393	165.85	0.0162	170.68	0.0178	166.57	0.0146
222.31	0.1339	165.18	0.0144	192.23	0.4090	181.63	0.4154	186.18	0.3801	183.11	0.4379
232.29	0.0707	183.41	0.4332	206.97	0.1419	212.34	0.1780	218.13	0.1506	215.98	0.1467
277.03	0.0221	215.52	0.1672	231.44	0.1467	242.06	0.0406	248.86	0.0382	245.60	0.0385
		247.57	0.0365	272.99	0.0288						
4-F-aniline											
E	f	E	f	E	f	E	f	E	f	E	f
146.36	0.0031	167.47	0.0600	190.53	0.2346	174.70	0.2661	176.95	0.1239	170.59	0.1098
218.88	0.1156	178.78	0.3663	195.05	0.1173	210.34	0.1570	182.48	0.3175	179.51	0.3779
227.74	0.0567	214.03	0.1560	231.99	0.1613	251.19	0.0545	217.30	0.1427	214.64	0.1298
279.66	0.0177	257.96	0.0516	289.10	0.0372			261.43	0.0495	255.29	0.0523
290.31	0.0129										
362.29	0.0011										
3-Cl-aniline											
E	f	E	f	E	f	E	f	E	f	E	f
203.56	0.5666	184.97	0.0972	211.31	0.3493	188.35	0.2962	196.40	0.2627	189.36	0.3286
220.06	0.0156	188.32	0.3447	229.08	0.0765	212.30	0.1087	219.84	0.0771	189.36	0.3286
234.55	0.1461	217.16	0.1188	281.54	0.0293	248.60	0.0512	257.60	0.0450	189.60	0.0936
284.16	0.0267	253.58	0.0482							217.53	0.0932
4MeOacetophenone											
E	f	E	f	E	f	E	f	E	f	E	f
197.60	0.2896	190.87	0.2276	241.21	0.0681	191.48	0.1769	214.30	0.0856	194.80	0.1665
216.87	0.1283	240.65	0.1964	282.30	0.1071	237.78	0.1983	250.42	0.1739	241.78	0.1865
224.69	0.0437	298.01	0.0034	334.28	0.0112	308.53	0.0023	305.07	0.0050	293.62	0.0044
265.81	0.0368										
267.15	0.2202										
283.86	0.1390										
369.32	0.0011										
1,3-dimethoxybenzene											
E	f	E	f	E	f	E	f	E	f	E	f
193.84	1.0740	159.93	0.0293	189.16	0.0403	160.78	0.0217	168.15	0.0695	160.56	0.0371
220.80	0.1107	187.38	0.5333	204.48	0.5099	184.98	0.5289	191.91	0.4615	186.65	0.5361
261.93	0.0096	211.64	0.0624	225.89	0.0915	207.53	0.0620	215.21	0.0482	212.29	0.0506
271.73	0.0127	244.33	0.0484	267.60	0.0384	237.43	0.0527	245.18	0.0486	241.47	0.0492
279.19	0.0029					611.96	0.0081				

A.2 Python Scripts

A.2.1 Preparing MD Input Files

This script does two things. First (line 35-84), it calculates appropriate radius for solvent boundary potential without empirically fitting it. A simple model is proposed: at most solute will rotate around its outmost solute atom. In the code, the distance between the outmost solute atom to the solute's CG is called solute radius. The distance between the outmost solvent atom to the solute's CG is solvent radius. These two radius plus an extra 2-3 Angstrom gives SSBP radius for MD input file. Second (line 87-155), the script parses xyz file's geometry data into MD input file. Slight format change from xyz file type is required for GAMESS input files, but this python code automate that change. The output file is MD file which can be run on GAMESS. Output of this script can be seen below in MD Input File section.

```
1  ############################################################
2  ### Create inp for MD run from xyz file from packmol     ###
3  ### ssbp radius is also calculated roughly from          ###
4  ### starting geometry                                    ###
5  ############################################################
6
7  import sys
8  import csv
9  import os
10 import string
11
12
13 #for asking what the input in terminal should be
14 try:
15         if str(sys.argv[1])=='?':
16                 print '\nCall function as: prepareMD.py input.xyz numberOfSoluteAtoms
                    ↪  numberofSolventAtoms numberOfSolventMolecules \n'
17                 sys.exit()
18 except IndexError:
19     print '\n!!!Input command Error. Call function as: prepareMD.py input.xyz numberOfSoluteAtoms
        ↪  numberofSolventAtoms numberOfSolventMolecules \n'
20     sys.exit()
21 #for assigning received input from terminal
22 try:
23     input=str(sys.argv[1])
24     numberofSoluteAtoms=int(sys.argv[2])
25     numberofSoventAtoms=int(sys.argv[3])
26     numberOfSolventMolecules=int(sys.argv[4])
27 except IndexError:
28     print '\n!!!Input command Error. Call function as: prepareMD.py input.xyz numberOfSoluteAtoms
        ↪  numberofSolventAtoms numberOfSolventMolecules \n'
29     sys.exit()
30 #generate output name
31 if input.endswith('.xyz'):
32     output = input[:-4]+'.inp'
33 #for safety - at worst the output will not overwrite the input
34 else:
35     output=input+'.inp'
36
37 #part one
```

```
#This part is for finding ssbp radius for inout file
#enumerate gets data in line - line and line index - n
radiusInSolute=0.0
radiusInSolvent=0.0
avgX=0.0
avgY=0.0
avgZ=0.0
X=[]
Y=[]
Z=[]

lineNumber=0
#open input
f2=open(input)
for line in f2:
        lineNumber+=1
        #first two line does not contain useful info - x y z start on the third line
        if lineNumber>2:
                #x y z
                lineSplit=line.split()
                X.append(float(lineSplit[1]))
                Y.append(float(lineSplit[2]))
                Z.append(float(lineSplit[3]))
#for looping through array below
size=len(X)
#find a CG for solute atoms
avgX=sum(X[:numberofSoluteAtoms-1])/numberofSoluteAtoms
avgY=sum(Y[:numberofSoluteAtoms-1])/numberofSoluteAtoms
avgZ=sum(Z[:numberofSoluteAtoms-1])/numberofSoluteAtoms
#looping to find radius of each atoms in relative to solute's CG
#also find the maximum value of them
for i in range(0,size):
                d=((X[i]-avgX)**2+(Y[i]-avgY)**2+(Z[i]-avgZ)**2)**0.5
                if i<numberofSoluteAtoms:
                        if radiusInSolute<d:
                                radiusInSolute=d
                else:
                        if radiusInSolvent<d:
                                radiusInSolvent=d

#radius should be a little bit larger than the two combined - 3 Angstrom larger - this does not
 ↪  need to be super accurate
radiusInSolute=radiusInSolute
radiusInSolvent=radiusInSolvent
ssbpRadius=radiusInSolute+radiusInSolvent+3
print '\n'
print 'Radius in solute is:\t'+str(radiusInSolute)
print 'Radius in solvent is:\t'+str(radiusInSolvent)
print 'ssbp Radius should be:\t'+str(ssbpRadius)
print '\n'
################################

#Part two - this is where geometry data is taken from xyz, change into GAMESS input's format +
 ↪  other input
numberOfAllSolventsAtoms=numberofSoventAtoms*numberOfSolventMolecules
fragmentNumber=1;
atomLabel=1

#this dict is for generating atomic number from Acronym
```

```
atomicNumber={'LV': 116.0, 'BE': 4.0, 'FR': 87.0, 'BA': 56.0, 'BH': 107.0, 'BI': 83.0, 'BK': 97.0,
    'EU': 63.0, 'FE': 26.0, 'BR': 35.0, 'ES': 99.0, 'FL': 114.0, 'FM': 100.0, 'RG': 111.0, 'RU':
    44.0, 'NO': 102.0, 'NA': 11.0, 'NB': 41.0, 'ND': 60.0, 'NE': 10.0, 'RE': 75.0, 'RF': 104.0,
    'LU': 71.0, 'RA': 88.0, 'RB': 37.0, 'NP': 93.0, 'RN': 86.0, 'RH': 45.0, 'B': 5.0, 'CO': 27.0,
    'TH': 90.0, 'CM': 96.0, 'CL': 17.0, 'H': 1.0, 'CA': 20.0, 'CF': 98.0, 'CE': 58.0, 'N': 7.0,
    'CN': 112.0, 'P': 15.0, 'GE': 32.0, 'GD': 64.0, 'GA': 31.0, 'V': 23.0, 'CS': 55.0, 'CR': 24.0,
    'DS': 110.0, 'CU': 29.0, 'SR': 38.0, 'UUP': 115.0, 'UUS': 117.0, 'TC': 43.0, 'KR': 36.0, 'SI':
    14.0, 'SN': 50.0, 'SM': 62.0, 'UUT': 113.0, 'SC': 21.0, 'SB': 51.0, 'TA': 73.0, 'OS': 76.0,
    'PU': 94.0, 'SE': 34.0, 'AC': 89.0, 'HS': 108.0, 'YB': 70.0, 'DB': 105.0, 'C': 6.0, 'HO':
    67.0, 'DY': 66.0, 'HF': 72.0, 'HG': 80.0, 'HE': 2.0, 'PR': 59.0, 'PT': 78.0, 'LA': 57.0, 'F':
    9.0, 'UUO': 118.0, 'LI': 3.0, 'PB': 82.0, 'TL': 81.0, 'TM': 69.0, 'LR': 103.0, 'PD': 46.0,
    'TI': 22.0, 'TE': 52.0, 'TB': 65.0, 'PO': 84.0, 'PM': 61.0, 'ZN': 30.0, 'AG': 47.0, 'NI':
    28.0, 'I': 53.0, 'K': 19.0, 'IR': 77.0, 'AM': 95.0, 'AL': 13.0, 'O': 8.0, 'S': 16.0, 'AR':
    18.0, 'AU': 79.0, 'AT': 85.0, 'W': 74.0, 'IN': 49.0, 'Y': 39.0, 'CD': 48.0, 'ZR': 40.0, 'ER':
    68.0, 'MD': 101.0, 'MG': 12.0, 'PA': 91.0, 'SG': 106.0, 'MO': 42.0, 'MN': 25.0, 'AS': 33.0,
    'MT': 109.0, 'U': 92.0, 'XE': 54.0}

#write out put the headers - all the commands for GAMESS + ssbp
#functional = M06-2X - DFTTYP=M06-2X
f = open(output, 'w');
f.write(''' $CONTRL SCFTYP=RHF RUNTYP=MD COORD=UNIQUE
    DFTTYP=CAMB3LYP MAXIT=200 ICHARG=0 MULT=1 $END
 $MD KEVERY=10 PROD=.T. NVTNH=2 MBT=.T. MBR=.T.
    BATHT=298 RSTEMP=.T. DTEMP=25 NSTEPS=50000
    SSBP=.T. SFORCE=1.0 DROFF='''+str(ssbpRadius)+''' $END
 $DFT DC=.F. $END
 $SYSTEM MWORDS=1000 MEMDDI=1000 $END
 $SCF DIRSCF=.T. $END
 $BASIS GBASIS=N31 NGAUSS=6 NDFUNC=2 NPFUNC=1
    DIFFS=.TRUE. POLAR=POPN31 $END
 $DATA\n'''+ 'MD INPUT for' +input+'\nC1 1\n''')

#geometry
with open(input) as f1:
    #read by line
    #readlines if okay to use bc xyz is not too big
    lines = f1.readlines()
    #enumerate gets data in line - line and line index - n
    for n, line in enumerate(lines):
        #take all solute molecules (in range of 2 (line 3 where packmol starts) to num+2)
        #it's num+2 bc the range will go to num+1
        if n == 2:
            print 'Now Writing Solute:\n'
        if n in range(2,numberofSoluteAtoms+2):
            lineSplit=line.split();
            lineSplit.insert(1,str(atomicNumber[lineSplit[0]]))
            #convert coordinates to 10 decimals (add zeros if need be)
            for index in [2,3,4]:
                lineSplit[index]=float(lineSplit[index])
                lineSplit[index]=format(lineSplit[index],'.10f')
                grandString=lineSplit[0]+'\t'+lineSplit[1]+'\t'+lineSplit[2] +
                    '\t'+lineSplit[3]+'\t'+lineSplit[4]+'\n';
            f.write(grandString)
            print grandString
        if n == numberofSoluteAtoms+2:
            f.write(' $END\n\n $EFRAG\nCOORD=CART  POSITION=OPTIMIZE\n')
            print 'Now Writing Solvent:\n'
        #now start doing solvent - (need to add fragment number and atom labels)
        startPointOfSolvent=numberofSoluteAtoms+2
```

```
        if n in range(startPointOfSolvent, startPointOfSolvent+numberOfAllSolventsAtoms+1):
            #atomlabel = O1, H2, H3 from O, H, H
            if atomLabel%numberofSoventAtoms==1:
                grandString='FRAGNAME=H2ODFT ! '+str(fragmentNumber)+'\n'
                f.write(grandString)
                print grandString
                fragmentNumber+=1;
                atomLabel%=numberofSoventAtoms
            lineSplit=line.split();
            lineSplit.insert(1,str(atomLabel))
            atomLabel+=1
            #convert coordinates to 10 decimals (add zeros if need be)
            for index in [2,3,4]:
                lineSplit[index]=float(lineSplit[index])
                lineSplit[index]=format(lineSplit[index],'.10f')
            grandString=' '+lineSplit[0]+lineSplit[1]+'\t'+lineSplit[2] +
             ↪   '\t'+lineSplit[3]+'\t'+lineSplit[4]+'\n';
            f.write(grandString)
            print grandString
    #close the inp with £END
    f.write(' $END\n')
################################################################
```

A.2.2 MD Geometries Extraction

One of the reasons, an MD run might fail is if solute molecule is pushed out of the water sphere. 3dExtract4.py allows geometries of the system at different time to be extracted from a large size log file into a xyz-movie file. xyz files, capable of containing more than one frame of geometries, allows one to follow MD through a combination of screenshot (each frame is 10 femtoseconds - in the current MD input file - see MD Input File section).

```
#############################################################
### 3dExtract pulls out geometries from MD run and make ###
### an xyz-movie file for inspection MD       progress            ###
#############################################################

import os as os
import sys

#for asking what the input in terminal should be
try:
        if str(sys.argv[1])=='?':
                print '\nCall function as: 3dExtract.py input.log numberOfSoluteAtoms
                 ↪  numberofSolventAtoms numberOfSolventMolecules   \n'
                sys.exit()
except IndexError:
    print '\n!!!Input command Error. Call function as: 3dExtract.py input.log numberOfSoluteAtoms
     ↪  numberofSolventAtoms numberOfSolventMolecules   \n'
    sys.exit()

#Call as 3dExtract.py inputfile #ofsoluteAtom #ofsolventAtom #ofsoluteMolecules
try:
    input=str(sys.argv[1])
    numberofSoluteAtoms=int(sys.argv[2])
```

```
    numberofSoventAtoms=int(sys.argv[3])
    numberOfSolventMolecules=int(sys.argv[4])
except IndexError:
    print '\n!!!Input command Error. Call function as: 3dExtract.py input.log numberOfSoluteAtoms numberofSolventAtoms numberOfSolventMolecules \n'
    sys.exit()
if input.endswith('.log'):
    output = str(input[:-4])+'.xyz'
else:
    output=str(input)

numberOfAllSolventsAtoms=numberofSoventAtoms*numberOfSolventMolecules
#This is for comparing files to be written
previousGrandString=''
collectionStarted=False
FoundFirstGeometry=False
foundTime=False
foundPE=False
time=''
energy='0'
grandString=''
lineBwTimeAndEnergy=0
lineBwTimeAndGeometry=0
#1 is for cartesian line (useless), then 1 in 3(n+1) is for fragment H2O line (also useless)
numberOfLinesToBecollected=numberofSoluteAtoms+1+numberOfSolventMolecules*(numberofSoventAtoms+1)


#number of molecules so far
timeCount=0
#total number of atoms (solute + solvent) - used later in checking if file is complete
atomCount=0
#define functions here
lineSinceTimeIsFound=0;
#do an input of solvent, solute atoms
molList=[]
#for finding PE
lineCountFromTime=0
#for printing time
def printAndReturnTime (thisLine):
    lineComponents=thisLine.split();
    timeString=str(lineComponents[3]);
    #get rid off .00
    timeString=timeString[:-3]
    print "Analyzing t = "+timeString+" fsec\n"
    return timeString

#to determine if line should be collected -
def shouldCollect():
    #only check if collection is in progress - if it is, then continue to finish collecting the lines
    #collectionStarted is determined when ' QM ATOM COORDINATES (ANG)'  is found
    if collectionStarted:
        #from first solute atom to the last fragment atom
        if (atomCount>=0 and atomCount<numberOfLinesToBecollected):
            return True;
        else:
            return False;
    else:
        return False;
```

```
# only write when atomCount==numberOfLinesToBecollected

def shouldWrite():
    #only check if collection is in progress
    if collectionStarted:
        #solute
        if (atomCount==numberOfLinesToBecollected):
            return True;
        else:
            return False;
    else:
        return False;

def shouldCollectTime(line):
    #only check if collection is in progress
    if ' *** AT T=' in line:
        return True;
    else:
        return False;

def shouldCollectPE(line,reference,currentLine):
    #check if line bw time and energy is known - this is written as reference
    if reference>0:
        if currentLine==reference:
            return True
        else:
            return False
    #if reference is not known, then it needs to be found by finding string POT EN...
    elif reference==0:
        if '     POT  ENERGY' in line:
            reference=int(currentLine)
            return True;
        else:
            return False;

def shouldStartCollectGeometry(line,reference,currentLine):
    #check if line bw time and geometry is known - reference
    if reference>0:
        if currentLine==reference:
            return True
        else:
            return False
    #if not then it needs to be found by searching for the string TEMPER...
    elif reference==0:
        if ' QM ATOM ' in line:
            reference=int(currentLine)
            return True;
        else:
            return False;

#clear output.xyz
f = open(output, 'w');
f.write('')
#open input
f1=open(input)
#enumerate gets data in line - line and line index - n
#readlines() is eliminated because it creates a huge array and python cannot handle it when log
 ↪  file get very large
#using for line in... alleviate the burden on memory and actually speed up the process
```

```
for line in f1:
    #this keyword is usually before coordinate
    #find out if we have found the first geoetry
    if not FoundFirstGeometry:
        #if not then find it
        if shouldCollect():
            #split line
            lineSplit=line.split()
            atomCount+=1;
            #append to molList
            molList.append(lineSplit)
            time=str(0)
            if shouldWrite():
                atomCount=atomCount-(numberOfSolventMolecules+1);
                #for loop through a ***COPY*** of molList and delete some element from molList!
                #if you don't realize six asterisk then you should go back up - we do this so we
                 ↪  can remove element along the way without messing up the index
                for line in list(molList):
                    #if line has 4 elements then it's a coordinate from solvent fragment - we have
                     ↪  to drop number behind atom - O1 to O
                    if len (line) == 4:
                        #store string
                        oldString =  line[0]
                        #replacement string
                        newString=''
                        #loop to check if it's a alphabet or not
                        for character in range(len(oldString)):
                            #do substring of 1 character
                            subString = oldString[character:character+1]
                            #check if it's an alphabet - yes? then add to newString
                            if subString.isalpha():
                                newString = newString + subString
                        #replace 'O1' with 'O'
                        line[0]=newString
                    #if it's 5 then it's solute coordinate - we have to get rid of atomic number
                     ↪  behind atomic representation
                    elif len(line) == 5:
                        # 'N 7.0 ...' will become 'N ...'
                        del line[1]
                    #the rest are crap - just remove it out of the line
                    else:
                        #there's a reason why this is remove - not del - since we are iterating if
                         ↪  we delete using index we are gonna be screwed
                        molList.remove(line)
                #this is for if we have an incomplete file or inconsistant number of atoms we
                 ↪  should only use the one before and break for loop without appending to
                 ↪  grandString
                if len(molList) != atomCount:
                    print 'error'
                #xyz file has a format that we need atomCount at the top followed by snapshot
                 ↪  number(timeCount) on the next line before adding any coordinates
                grandString=grandString+str(atomCount)+'\n'+str(timeCount)+'\tfs\t'+str(energy)+'
                 ↪  KCAL/MOL\n'
                f.write(grandString)
                #loop tho molList to add data - molList = [['N','1','1','1'],['C'...],... ] And
                 ↪  element = ['N','1','1','1']
                for element in molList:
                    #loop through element in molList data = 'N','1','1','1'
                    for data in element:
```

```
                        #add to grandString and don't forget tab, return
                        f.write(data+'\t')
                    #end one element with a return
                    f.write('\n')
                #reset all values after writing
                atomCount=0;
                molList=[]
                collectionStarted=False;
                FoundFirstGeometry=True;
                timeCount+=1
        if ' QM ATOM COORDINATES (ANG)' in line:
            collectionStarted=True

    if FoundFirstGeometry:
        if not foundTime:
            if shouldCollectTime(line):
                time=printAndReturnTime(line);
                foundTime=True;
                if time=='0.00':
                    foundTime=False
                    grandString=''
        elif not foundPE:
            if shouldCollectPE(line,lineBwTimeAndEnergy,lineCountFromTime):
                #split line using space - sample(     POT  ENERGY      =     -1.804578585E+05
                 ↪  KCAL/MOL)
                #this will be split to [..., '=', '-1.804578585E+05'...] -time = element 4
                lineComponents=line.split();
                energy=str(lineComponents[3])
                foundPE=True
        elif shouldCollect():
            #split line
            lineSplit=line.split()
            atomCount+=1;
            #append to molList
            molList.append(lineSplit)
            if shouldWrite():
                atomCount=atomCount-(numberOfSolventMolecules+1);
                #for loop through a ***COPY*** of molList and delete some element from molList!
                #if you don't realize six asterisk then you should go back up - we do this so we
                 ↪  can remove element along the way without messing up the index
                for line in list(molList):
                    #if line has 4 elements then it's a coordinate from solvent fragment - we have
                     ↪  to drop number behind atom - O1 to O
                    if len (line) == 4:
                        #store string
                        oldString =  line[0]
                        #replacement string
                        newString=''
                        #loop to check if it's a alphabet or not
                        for character in range(len(oldString)):
                            #do substring of 1 character
                            subString = oldString[character:character+1]
                            #check if it's an alphabet - yes? then add to newString
                            if subString.isalpha():
                                newString = newString + subString
                        #replace 'O1' with 'O'
                        line[0]=newString
                    #if it's 5 then it's solute coordinate - we have to get rid of atomic number
                     ↪  behind atomic representation
```

```
242                     elif len(line) == 5:
243                         # 'N 7.0 ...' will become 'N ...'
244                         del line[1]
245                     #the rest are crap - just remove it out of the line
246                     else:
247                         #there's a reason why this is remove - not del - since we are iterating if
                             ↪  we delete using index we are gonna be screwed
248                         molList.remove(line)
249                 #this is for if we have an incomplete file or inconsistant number of atoms we
                     ↪  should only use the one before and break for loop without appending to
                     ↪  grandString
250                 if len(molList) != atomCount:
251                     print 'error'
252                 #xyz file has a format that we need atomCount at the top followed by snapshot
                     ↪  number(timeCount) on the next line before adding any coordinates
253                 grandString=grandString+str(atomCount)+'\n'+str(time)+'\tfs\t'+str(energy)+'
                     ↪  KCAL/MOL\n'
254                 f.write(grandString)
255                 #loop tho molList to add data - molList = [['N','1','1','1'],['C'...],... ] And
                     ↪  element = ['N','1','1','1']
256                 for element in molList:
257                     #loop through element in molList data = 'N','1','1','1'
258                     for data in element:
259                         #add to grandString and don't forget tab, return
260                         f.write(data+'\t')
261                     #end one element with a return
262                     f.write('\n')
263                 #reset all values after writing
264                 atomCount=0;
265                 molList=[]
266                 collectionStarted=False;
267                 foundTime=False
268                 foundPE=False
269                 lineCountFromTime=0;
270                 grandString=''
271                 timeCount+=1
272         elif shouldStartCollectGeometry(line,lineBwTimeAndGeometry,lineCountFromTime):
273                 collectionStarted=True
274     lineCountFromTime+=1
275 f.close()
276 #sanity check
277 print 'Done. Extract ' + str(timeCount) + ' snapshots total.'
278
```

A.2.3 Plot Potential Energy Of MD Run

plotEnergyMD6.py script extracts potential energy and temperature of each MD frame to determine the if the system is close to equilibration. This script and 3dExtract are very essential to the first stage of the project: they determine whether MD has failed or reached equilibrium based on the geometry and potential energy of the system. Like 3dExtract, many versions of this code has been developed and modified and they are the most refined pieces of code for their purpose. In the future when the codes are unified, improvement can be made on plotting the plot on Matlab instead of matplotlib.

```
#############################################################
### Use this to plot energy vs time to see if MD has    ###
### run its course. Generates: csv of PE and            ###
### temperature vs time, pdf of the plot                ###
#############################################################

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import csv
import sys
import string

#call as plotEnergyMD3.py inputfile
#for asking what the input in terminal should be
try:
        if str(sys.argv[1])=='?':
                print '\nCall function as: plotEnergyMD.py input.log   \n'
                sys.exit()
        else:
                input=str(sys.argv[1])
except IndexError:
    print '\n!!!Input command Error. Call function as: plotEnergyMD.py input.log \n'
    sys.exit()
output=str(input) + '_energies.csv'

#initiate variables
lineBwTimeAndEnergy=0;
lineBwTimeAndTemp=0;
lineCountFromTime=0;
collectionStarted=False
foundTime=False
foundPE=False
foundTemp=False
firstTime=True
grandString=''
time='0'

################################
#functions

#check if time is in line
def shouldCollectTime(line):
    #only check if collection is in progress
    if ' *** AT T=' in line:
        return True;
    else:
        return False;

# for printing time so one can keep track of the progress
def printTime (thisLine):
    lineComponents=thisLine.split();
    timeString=str(lineComponents[3]);
    print "Analyzing t = "+timeString+" fsec\n"

#Are we currently looking potential energy?
def shouldCollectPE(line,reference,currentLine):
    #check if line bw time and energy is known - this is written as reference
    if reference>0:
```

```
60          if currentLine==reference:
61              return True
62          else:
63              return False
64      #if reference is not known, then it needs to be found by finding string POT EN...
65      elif reference==0:
66          if '     POT  ENERGY' in line:
67              reference=int(currentLine)
68              return True;
69          else:
70              return False;
71  #Are we currently looking Temp?
72  def shouldCollectTemp(line,reference,currentLine):
73      #check if line bw time and energy is known - reference
74      if reference>0:
75          if currentLine==reference:
76              return True
77          else:
78              return False
79      #if not then it needs to be found by searching for the string TEMPER...
80      elif reference==0:
81          if '     TEMPER' in line:
82              reference=int(currentLine)
83              return True;
84          else:
85              return False;
86
87  #once everything is found, we should write down before moving on to the next snapshot
88  def shouldWrite():
89      #only check if collection is in progress
90      if foundTime:
91          if foundPE:
92              if foundTemp:
93                  return True;
94              else:
95                  return False
96          else:
97              return False;
98      else:
99          return False;
100
101 ################################
102 #open csv and prepare for writing
103 f = open(output, 'w');
104 f.write('')
105
106 #open input
107 f1=open(input)
108 print 'finding patterns...'
109 #avoid using readlines() so there'll be no problem with large files
110 for line in f1:
111     #time keyword is before PE, PE is before Temp so the search should be in this order in order to
        ↪  be most efficient
112     #find out if collection is needed
113     if not foundTime:
114         if shouldCollectTime(line):
115             printTime(line)
116             #split line
117             lineComponents=line.split();
```

```
118             #append time (split) to string
119             #split line using space - sample(*** AT T=         10.00 FSEC, THIS RUN'S STEP NO.=
                ↪  10)
120             #this will be split to ['***', 'AT', 'T=', '10.00'...] -time = element 4
121             newTime=str(lineComponents[3])[:-3]
122             if int(time)>=int(newTime):
123                     print 'there is a duplication at '+time+' and '+newTime +' ignoring new values'
124             else:
125                             time=str(newTime)
126                             grandString=grandString+time;
127                             foundTime=True;
128     elif not foundPE:
129         if shouldCollectPE(line,lineBwTimeAndEnergy,lineCountFromTime):
130             #append time (split) to string
131             #split line using space - sample(     POT  ENERGY      =     -1.804578585E+05 KCAL/MOL)
132             #this will be split to [..., '=', '-1.804578585E+05'...] -time = element 4
133             lineComponents=line.split();
134             grandString=grandString+','+str(lineComponents[3])
135             foundPE=True
136     elif not foundTemp:
137         if not firstTime:
138             if shouldCollectTemp(line,lineBwTimeAndTemp,lineCountFromTime):
139                 #append time (split) to string
140                 #split line using space - sample(     TEMPERATURE      =         349.98666547 K)
141                 #this will be split to [..., '=', '-349.98666547'...] -time = element 3
142                 lineComponents=line.split();
143                 grandString=grandString+', '+str(lineComponents[2])+'\n';
144                 foundTemp=True
145         #problem with this is - the first snapshot's temperature is not given in the log file
146         #set Temp to 0 to indicate the beginning
147         if firstTime:
148             grandString=grandString+',0\n';
149             #once append, turn off the boolean
150             firstTime=False
151             foundTemp=True
152     #write after all data is collected for one snapshot
153     if shouldWrite():
154         with open(output, 'a') as f:
155             f.write(grandString)
156         #reset the variables
157         lineCountFromTime=0;
158         collectionStarted=False
159         foundTime=False
160         foundPE=False
161         foundTemp=False
162         grandString=''
163     lineCountFromTime+=1;
164 #finish writing csv
165 f.close()
166
167 #plot
168 #pull out CSV
169 #use csv.reader bc csv has ',' and this automate the formatting
170 f = csv.reader(open(output))
171 #convert column to array using zip (a built in function)
172 Time, Energy, Temp = zip(*f)
173 #convert string to float
174 Time = map(float, Time)
175 Energy = map(float, Energy)
```

```
Temp = map(float, Temp)

minEnergy=min(Energy)
relativeEnergy=[]
for eachEnergy in Energy:
        relativeEnergy.append(eachEnergy-minEnergy)

#plot
x = Time
y1 = Energy
y2 = Temp

plt.subplot(2, 1, 1)
plt.plot(x, y1, 'g-')
plt.title('MD PE and T plot of: ' + str(input))
plt.ylabel('Potential Energy (KCal/mol)')

plt.subplot(2, 1, 2)
plt.plot(x, y2, 'b-')
plt.xlabel('time (fs)')
plt.ylabel('System Temperature (K)')
plt.savefig(str(input) + '_EnergyPlot.pdf', format='pdf')

plt.gcf().clear()
y1 = relativeEnergy

plt.subplot(2, 1, 1)
plt.plot(x, y1, 'g-')
plt.title('MD relative PE and T plot of: ' + str(input))
plt.ylabel('Relative Potential Energy (KCal/mol)')

plt.subplot(2, 1, 2)
plt.plot(x, y2, 'b-')
plt.xlabel('time (fs)')
plt.ylabel('System Temperature (K)')
plt.savefig(str(input) + '_EnergyPlot_2.pdf', format='pdf')
```

A.2.4 Find The Most Equilibrated Period

There are currently no consensus as to how to determine if a system has reached the equilibrium in molecular dynamics. In previous works, plotEnergyMD (previous script) was used to indicate whether the potential energy of the system (solute and solvent) has stabilized. Arbitrariness in determining the equilibrium falls in the hands of users. findEquilibrium.py is designed to solve this subjectivity. With a list of potential energies at different time from plotEnergyMD, linear fit can be done in a fix interval to evaluate the rise or fall in energy. Currently, the limit value is taken, still empirically, from 15000 to 25000 fs interval in CAM-B3LYP aniline32.log. The lowest slope of the list is used to identify good range for excited state energy calculation. Further improvement can be done to find the bottom slope limit as a variable with molecule input.

```
############################################################
### This is used to determined the equilibrium using    ###
### linear regression and an upper limit for the slope        ###
### The set range for this script is 10 ps                  ###
```

```
#############################################################

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import csv
import sys
import numpy as np

#for asking what the input in terminal should be
try:
    if str(sys.argv[1])=='?':
        print '\nCall function as: findEquilibrium.py input.log   \n'
        sys.exit()
    else:
        input=str(sys.argv[1])
except IndexError:
    print '\n!!!Input command Error. Call function as: findEquilibrium.py input.log \n'
    sys.exit()
output=str(input) + '_energies.csv'
#This portion is the same as in plotEnergyMD6
#pull out CSV
f = csv.reader(open(output))
#convert column to array using zip (a built in function)
Time, Energy, Temp = zip(*f)
#convert string to float
Time = map(float, Time)
Energy = map(float, Energy)
Temp = map(float, Temp)

#find Equilibrium using linear regression
#this number control the range of time to be used in energy fluctuation calculation
minNumberOfStep=1000
#This is the limit above which the script will report no equilibrium is found
#this is from aniline32.log - 15000 to 25000
maxSlope=1e-4
#for plotting
slope=[]
print 'Finding equilibrium using minimum number of steps = '+str(minNumberOfStep) +' and top limit
 ↪  of acceptable slope = '+str(maxSlope)
try:
        #for looping
    size=len(Time)
    x=Time[0:minNumberOfStep]
    y=Energy[0:minNumberOfStep]
    #for using in loop
    #set thio a high value - it can be any number bc we will replace it with the lowest slope value
     ↪  found in loop
    lowestSlopeValue=1e5
    #Same - this will be replaced
    indexOfLowestSlope=-1
    for i in range(0,size-1-minNumberOfStep):
        print 'Finding equilibrium from t= '+str(Time[i])+' to '+str(Time[i+minNumberOfStep-1])
        #poly fit is basically a linear fit - m=slope, b=y_intersect
        m,b = np.polyfit(x, y, 1)
        #for plotting - append to array of existing slope values
        slope.append(m)
        print 'slope = ' +str(m)
        #take absolute value and see which interval does not fluctuate the least
```

```
        if abs(m)<=abs(lowestSlopeValue):
            lowestSlopeValue=float(m)
            indexOfLowestSlope=int(i)
        #this is similar to queue structure - room for improvement is to make x and y arrays into
         ↪  actual queues
        #else remove the head and add next tail
        del x[0]
        x.append(Time[minNumberOfStep+i])
        del y[0]
        y.append(Energy[minNumberOfStep+i])
#if x or y does not have enough element (minNumberOfStep) then report error
#room for improvement - move this up top instead of having a long try
except IndexError:
        print 'There is not enough data to determine the equilibrium'
#if lowestSlopeValue pass the top limit then report
if abs(lowestSlopeValue)<=abs(maxSlope):
        report=str('Found best equilibrium starting from '+str(Time[indexOfLowestSlope])+' to
         ↪  '+str(Time[indexOfLowestSlope+minNumberOfStep-1])+' with slope =
         ↪  '+str(lowestSlopeValue))
        print report
#if not then say so
else:
        report = 'Equilibrium is not yet reach'
        print report
        print str('The current limit is at '+str(maxSlope) +' kcal/mol/fs and the lowest value of
         ↪  slope = '+str(lowestSlopeValue))

#plot slope vs time
x=slope
#align time with slope
y=Time[0:size-1-minNumberOfStep]
plt.plot(y,x)
plt.gca().set_position((.125, .2, .8, .7))
plt.xlabel('Starting Time (fs)')
plt.ylabel('Slope (KCal/mol/fs)')
plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2), useOffset=False)
plt.title('Slope vs Time for the MD File: ' + str(input))
plt.figtext(.01, .05,report)
plt.savefig(str(input) + '_SlopePlot.pdf', format='pdf')
```

A.2.5 Prepare TD-DFT Input

After equilibrium is determined, fincut2.py can be used to create TD-DFT input files from xyz-movie file and a text file containing gmssub commands especially for Bowdoin HPC grid. The script was originally created by Ricke '14 for this work, but many improvement has been made. The updated script works faster and more efficient, even though it still has outdated syntax and methods.

```
############################################################
### This script create a folder of inp for TDDFT with   ###
### user-defined time range and TDDFT functional        ###
############################################################

import os
import sys

```

```
#call as fincut.py input.log startTime stopTime timePerFrame
#this will run from starting startTime+timePerFrame to stopTime
#for example 15010-25000 if input is 15000, 25000

#this dict is for generating atomic number from Acronym
atomicNumber={'LV': 116.0, 'BE': 4.0, 'FR': 87.0, 'BA': 56.0, 'BH': 107.0, 'BI': 83.0, 'BK': 97.0,
 ↪  'EU': 63.0, 'FE': 26.0, 'BR': 35.0, 'ES': 99.0, 'FL': 114.0, 'FM': 100.0, 'RG': 111.0, 'RU':
 ↪  44.0, 'NO': 102.0, 'NA': 11.0, 'NB': 41.0, 'ND': 60.0, 'NE': 10.0, 'RE': 75.0, 'RF': 104.0,
 ↪  'LU': 71.0, 'RA': 88.0, 'RB': 37.0, 'NP': 93.0, 'RN': 86.0, 'RH': 45.0, 'B': 5.0, 'CO': 27.0,
 ↪  'TH': 90.0, 'CM': 96.0, 'CL': 17.0, 'H': 1.0, 'CA': 20.0, 'CF': 98.0, 'CE': 58.0, 'N': 7.0,
 ↪  'CN': 112.0, 'P': 15.0, 'GE': 32.0, 'GD': 64.0, 'GA': 31.0, 'V': 23.0, 'CS': 55.0, 'CR': 24.0,
 ↪  'DS': 110.0, 'CU': 29.0, 'SR': 38.0, 'UUP': 115.0, 'UUS': 117.0, 'TC': 43.0, 'KR': 36.0, 'SI':
 ↪  14.0, 'SN': 50.0, 'SM': 62.0, 'UUT': 113.0, 'SC': 21.0, 'SB': 51.0, 'TA': 73.0, 'OS': 76.0,
 ↪  'PU': 94.0, 'SE': 34.0, 'AC': 89.0, 'HS': 108.0, 'YB': 70.0, 'DB': 105.0, 'C': 6.0, 'HO':
 ↪  67.0, 'DY': 66.0, 'HF': 72.0, 'HG': 80.0, 'HE': 2.0, 'PR': 59.0, 'PT': 78.0, 'LA': 57.0, 'F':
 ↪  9.0, 'UUO': 118.0, 'LI': 3.0, 'PB': 82.0, 'TL': 81.0, 'TM': 69.0, 'LR': 103.0, 'PD': 46.0,
 ↪  'TI': 22.0, 'TE': 52.0, 'TB': 65.0, 'PO': 84.0, 'PM': 61.0, 'ZN': 30.0, 'AG': 47.0, 'NI':
 ↪  28.0, 'I': 53.0, 'K': 19.0, 'IR': 77.0, 'AM': 95.0, 'AL': 13.0, 'O': 8.0, 'S': 16.0, 'AR':
 ↪  18.0, 'AU': 79.0, 'AT': 85.0, 'W': 74.0, 'IN': 49.0, 'Y': 39.0, 'CD': 48.0, 'ZR': 40.0, 'ER':
 ↪  68.0, 'MD': 101.0, 'MG': 12.0, 'PA': 91.0, 'SG': 106.0, 'MO': 42.0, 'MN': 25.0, 'AS': 33.0,
 ↪  'MT': 109.0, 'U': 92.0, 'XE': 54.0}

#if not sure use ? to ask
try:
        if str(sys.argv[1])=='?':
                print '\nCall function as: fincut.py input.log numberOfSoluteAtoms
                 ↪  numberofSolventAtoms numberOfSolventMolecules startTime stopTime timePerFrame
                 ↪  Functional\n'
                sys.exit()
except IndexError:
    print '\n!!!Input command Error. Call function as: fincut.py input.log numberOfSoluteAtoms
     ↪  numberofSolventAtoms numberOfSolventMolecules startTime stopTime timePerFrame
     ↪  Functional\n'
    sys.exit()

#Call as fincut.py input.log numberOfSoluteAtoms numberofSolventAtoms numberOfSolventMolecules
 ↪  startTime stopTime timePerFrame
try:
    input=str(sys.argv[1])
    numberOfSoluteAtoms=int(sys.argv[2])
    numberOfSolventAtoms=int(sys.argv[3])
    numberOfSolventMolecules=int(sys.argv[4])
    startTime =int(sys.argv[5])
    stopTime =int(sys.argv[6])
    timePerFrame=int(sys.argv[7])
    Functional=str(sys.argv[8])
except IndexError:
    print '\n!!!Input command Error. Call function as: fincut.py input.log numberOfSoluteAtoms
     ↪  numberofSolventAtoms numberOfSolventMolecules startTime stopTime timePerFrame
     ↪  Functional\n'
    sys.exit()

#LCBLYP has special file format
if Functional='LCBLYP':
        FunctionalPrint='BLYP'
        LC=true
else:
        FunctionalPrint=Functional
        LC=false
```

```
46
47
48  #should collect data when 1) time within interval specified
49  #2) they are x y z line - not line 1 and 2
50  def shouldCollect(numberOfLine):
51      frame=int(numberOfLine/totalNumberOfLineInOneSet)
52      time=frame*timePerFrame
53      #if within range
54      if time>startTime and time<=stopTime:
55          #0 and 1 (line 1 and 2) in each frame in xyz are not useful
56          if numberOfLine%totalNumberOfLineInOneSet!=0 and numberOfLine%totalNumberOfLineInOneSet!=1:
57              return True
58          else:
59              return False
60      else:
61          return False
62
63  #condition for writing to each inp
64  def shouldWrite(numberOfLine):
65      #should write when last line of a frame is read
66      if numberOfLine%totalNumberOfLineInOneSet==totalNumberOfLineInOneSet-1 :
67          return True
68      else:
69          return False
70
71  #write FRAGNAME=H2ODFT ! 1
72  def shouldWriteFragmentHeader(numberOfLineInSolvent):
73      #should write before writing geometry of solvent molecules
74      if numberOfLineInSolvent%numberOfSolventAtoms==0 :
75          return True
76      else:
77          return False
78
79  #create format of GAMESS inp
80  def insertAtomicNumberInto(line):
81                  lineSplit=line.split()
82                  lineSplit.insert(1,str(atomicNumber[lineSplit[0]]))
83                  return
                     ↪  lineSplit[0]+'\t'+lineSplit[1]+'\t'+lineSplit[2]+'\t'+lineSplit[3]+'\t'+lineSplit[4]+'\n'
84  #create format of GAMESS inp O -> O1, H-> H2...
85  def insertCountsInto(line, count):
86                  lineSplit=line.split()
87                  lineSplit[0]=lineSplit[0]+str(count)
88                  return lineSplit[0]+'\t'+lineSplit[1]+'\t'+lineSplit[2]+'\t'+lineSplit[3]+'\n'
89
90  #even tho input is received written in .log - it will ultimately use .xyz created by 3dExtract for
     ↪  efficiency
91  #this can be confusing - room for improvement
92  if input.endswith('.log'):
93      input = input[:-4]
94
95  #Path for storing input files
96  path = os.getcwd()
97  inputPath=path+'/'+input+'InputFiles_'+Functional
98  #create folders if not already done
99  if not os.path.exists(inputPath): os.makedirs(inputPath)
100
101 #initiate variables
102 grandString=[]
```

```
lineNumber=0
numberOfAllSolventsAtoms=numberOfSolventAtoms*numberOfSolventMolecules
totalNumberOfAtoms=numberOfSoluteAtoms+numberOfAllSolventsAtoms
totalNumberOfLineInOneSet=totalNumberOfAtoms+2

#if no xyz in folder - ask for it
    #if no 3dExtract.py - print...
    #if there is, call it from here + print s'th
#else create one?
#if there is, then proceed
#this is very confusing - if there is xyz but it's not updated when it's of no use - room for
 ↪  improvement

#open input.xyz
try:
    f=open(input+'.xyz')
except IOError:
    print '\nThere is currently no xyz file named '+input
    print 'trying to call 3dExtract'
    try:
        os.system('python 3dExtract4.py'+' '+input+'.log'+' '+str(numberOfSoluteAtoms)+'
         ↪  '+str(numberOfSolventAtoms)+' '+str(numberOfSolventMolecules))
    except IOError:
        print '\n  Error. There is no 3dExtract to call. Try copying 3dExtract here. \n'
        print 'Process terminated abnormally'
        sys.exit()
    f=open(input+'.xyz')

#since we are reading from xyz - using readlines() is okay
numberOfAllAtoms=int(f.readline().strip())
#enumerate gets data in line - line and line index - n
for line in f:
    lineNumber+=1
    #check if the line should be writen
    if shouldCollect(lineNumber):
        grandString.append(line)
        #if about to write -create file with this name
        if shouldWrite(lineNumber):
            frame=int(lineNumber/totalNumberOfLineInOneSet)
            time=frame*timePerFrame
            f1 = open(inputPath+'/'+input+'_'+str(time)+'.inp','w')
            #write header
            headerString=""" $CONTRL SCFTYP=RHF TDDFT=EXCITE DFTTYP="""+Functional+"""
       RUNTYP=ENERGY ICHARG=0 MULT=1 COORD=UNIQUE
       MAXIT=200 NPRINT=-5 ISKPRP=1 $END



 !TDDFT requires lots of memory space
 $SYSTEM MWORDS=200 MEMDDI=250 $END
 $SCF DIRSCF=.T. $END
 $TDDFT NSTATE=5 TPA=.f. $END\n"""
    if LC=false:
            headerString=headerString+' $DFT LC=.t. $END'
    else:
            headerString=headerString+' $DFT LC=.f. $END'
headerString=headerString+"""\n $BASIS GBASIS=N311 NGAUSS=6 NDFUNC=2 NPFUNC=1
       DIFFSP=.TRUE. DIFFS=.TRUE. POLAR=POPN311 $END
 $DATA\n"""+input+' at t= '+str(time)+'\nC1 1\n'
```

```
            print 'Making input file for '+input+' at t= '+str(time) +' with '+FunctionalPrint
            #write the header
            f1.write(headerString)
            #write one line by one - before writing we need to add atomic number in using the
             ↪  function defined above
            for eachLine in grandString[:numberOfSoluteAtoms]:
                eachLine=insertAtomicNumberInto(eachLine)
                f1.write (eachLine)
            #end solute and go to solvent
            stringBwSoluteAndSolvent=""" $END\n\n $EFRAG\nCOORD=CART  POSITION=OPTIMIZE \n"""
            f1.write(stringBwSoluteAndSolvent)
            #write fragments
            #for iteration
            i=0
            for numberOfLineWithInSolvents, eachLine in
             ↪  enumerate(grandString[numberOfSoluteAtoms:]):
                    #is header needed - if so write
                if shouldWriteFragmentHeader(numberOfLineWithInSolvents):
                    fragmentsNumber=numberOfLineWithInSolvents/numberOfSolventAtoms+1
                    f1.write("FRAGNAME=H2ODFT ! "+str(fragmentsNumber)+'\n')
                #write each line in fragments
                atomNumberInFragments=i%numberOfSolventAtoms+1
                eachLine=insertCountsInto(eachLine, atomNumberInFragments)
                f1.write (eachLine)
                i+=1
            #close and reset
            f1.write (" $END\n")
            grandString=[]
            f1.close()

#create megaio for gmssub input
allFiles = os.listdir(inputPath)
f = open('megaio_'+input+'_hpc.txt','w')
program = 'gmssub'
processors = '32'

#write item
for item in allFiles:
        outputName=str(item)[:-4]+'.log'
        f.write(program + ' %s '%item + processors +' '+outputName+' -l virtual_free=4g\n')

#for moving megaio to current folder and set permission to read
cmdstring = 'chmod 777 megaio_'+input+'_hpc.txt\n'+'mv megaio_'+input+'_hpc.txt '+inputPath
os.system(cmdstring)

#call for xsedeGenerator.py
#os.system('python xsedeShellGenerator.py '+input+'InputFiles_'+Functional)
```

A.2.6 Pull Excited State Energies From TD-DFT Log Files

postMDDataPull2.py pulls out excited state energies and dipole moments. Energy output is in the format of time, S1, S2... Dipole output is in the format of time, X1, Y1, Z1, X2...

```
############################################################
### Use this pull out excited state energies and dipole ###
### moments from all the files in specified folder      ###
### Output is time, S1, S2... or time, X1, Y1, Z1, X2...###
```

```
5  ############################################################
6
7  import os as os
8  import numpy as numpy
9  import matplotlib.pyplot as plt
10 import sys
11
12 #if not sure use ? to ask
13 try:
14         if str(sys.argv[1])=='?':
15                 print '\nCall function as: postMDDataPull2.py inputDirectory
                    ↪  NumberOfExcitedStateEnergies\n'
16                 sys.exit()
17 except IndexError:
18     print '\n!!!Input command Error. Call function as: postMDDataPull2.py inputDirectory
        ↪  NumberOfExcitedStateEnergies\n'
19     sys.exit()
20
21 #Call as fincut.py input.log numberOfSoluteAtoms numberofSolventAtoms numberOfSolventMolecules
    ↪  startTime stopTime timePerFrame
22 try:
23     input=str(sys.argv[1])
24     numberOfExcitedStates=int(sys.argv[2])
25 except IndexError:
26     print '\n!!!Input command Error. Call function as: postMDDataPull2.py inputDirectory
        ↪  NumberOfExcitedStateEnergies\n'
27     sys.exit()
28 # to prevent / at the end of the input file if used terminal autofill
29 if input.endswith('/'):
30     input = input[:-1]
31
32 #make input path - folder containing the log files or out files
33 path = os.getcwd()
34 inputPath=path+'/'+input
35
36 #arrays for storing and manipulating the data extracted
37 outputList=[]
38 fileList=os.listdir(inputPath)
39 listAllEAndf=[]
40 energies=[]
41 oscillatorStrengths=[]
42 timeList=[]
43 energyFinal=numpy.array(['Energy(eV)'])
44 oscillatorStrengthFinal=numpy.array(['Oscillator Strength'])
45
46 #get all the file names
47 for fileName in fileList:
48     if (".out" in fileName or ".log" in fileName):
49         outputList.append(fileName)
50
51 #open 2 output files
52 f1=open(input+'MD_data.csv','w')
53 f2=open(input+'MD_dipole.csv','w')
54 #now read each file and collect data
55 for fileName in outputList:
56         eachFile=inputPath+'/'+fileName
57         f=open(eachFile,'r')
58         lines=f.readlines()
59         startingLine=0
```

```
        fileIsComplete=False
        #find if the file is complete
        # use enumerate and reverse
        for i, line in reversed(list(enumerate(lines))):
                if ' STATE #   1  ENERGY =    ' in line:
                        startingLine=i
                        fileIsComplete=True
                        break
        # now start collecting data
        if fileIsComplete:
                print 'Harvesting data from file named: '+fileName
                counter=0
                #loop
                for n, line in enumerate(lines):
                        #time
                        if line.startswith('     RUN TITLE'):
                                nextLine=lines[n+2]
                                time=str(nextLine.split()[3])
                                f1.write(time)
                                f2.write(time)
                        #energy
                        if line.startswith(' STATE # '):
                                E=line.split()[5]
                                nextLine=lines[n+1]
                                f=nextLine.split()[3]
                                f1.write(','+E+','+f)
                                counter+=1
                        #stop collecting
                        if (counter==numberOfExcitedStates):
                                f1.write('\n')
                                counter=0
                        #dipole
                        if line.startswith('                          SUMMARY OF TDDFT RESULTS'):
                                #5 lines are from 'SUMMARY...' to ' 1  A ...' which starts to
                                 ↪  contain dipole moments
                                for linesAhead in range(5,5+numberOfExcitedStates):
                                        lineContainingDipoles=lines[n+linesAhead]
                                        lineSplit=lineContainingDipoles.split()
                                        #locations of dipole in line
                                        [X,Y,Z]=[lineSplit[4], lineSplit[5], lineSplit[6]]
                                        f2.write(','+X+','+Y+','+Z)
                                f2.write('\n')
```

A.3 GAMESS Inputs

A.3.1 MD Input File

MD run is core to modeling explicit solvent. The MD run is simulated every femtosecond but only record every 10 femtoseconds. The bath temperature is 25 $\pm$ 25 degree Celsius. Solvent boundary potential is also activated using default Sforce value ($1.0 kcal \cdot mol^{-1} \cdot A^{-2}$), but with estimate ssbp radius. ########n######### are for restarting MD in case the calculation abruptly ends (see next section). In this version, dispersion correction is not turned on. Basis set = 6-31+G(2d,p)

```
 # run type = MD, with functional = CAMB3LYP, COORD = UNIQUE is important
 $CONTRL SCFTYP=RHF RUNTYP=MD COORD=UNIQUE
    DFTTYP=CAMB3LYP MAXIT=200 ICHARG=0 MULT=1 $END
 # MD is recording every 10 frames with default 1 frame =10 fs
 # 25 degree celcius, RSTEMP is on for keeping the temp ~ +/-25
 # ssbp is on with default SForce value and radius estimated from prepareMD2.py
 $MD KEVERY=10 PROD=.T. NVTNH=2 MBT=.T. MBR=.T.
    BATHT=298 RSTEMP=.T. DTEMP=25 NSTEPS=50000
    SSBP=.T. SFORCE=1.0 DROFF=12.0632116659 $END
 #########1#########
 ####################
 # dispersion correction is off
 $DFT DC=.F. $END
 # memory requested at each node =1000 million words
 # memory reserved for communication = 1000 million words
 $SYSTEM MWORDS=1000 MEMDDI=1000 $END
 $SCF DIRSCF=.T. $END
 # Basis set = 6-31+(2d,p)
 $BASIS GBASIS=N31 NGAUSS=6 NDFUNC=2 NPFUNC=1
    DIFFS=.TRUE. POLAR=POPN31 $END
 # solute geometry - C1 1 = symmetry data
 $DATA
MD INPUT for aniline32
C1 1
 #########2#########
N        7.0        -2.3128100000        -0.0046000000        -0.0894530000
C        6.0        -0.9197160000        -0.0031280000        -0.0360090000
C        6.0        -0.2076150000        1.2004070000        -0.0355880000
.
.
H        1.0        -2.7544730000        0.8437050000        0.2369240000
H        1.0        -2.7570450000        -0.8248640000        0.2993670000
 ####################
 $END

 # solvent geometry in EFP1 (EFP2 is still not available)
 #########3#########
 $EFRAG
COORD=CART  POSITION=OPTIMIZE
FRAGNAME=H2ODFT ! 1
 O1        1.7760990000        4.8390610000        -2.1049530000
 H2        0.9224740000        4.4173920000        -2.2628490000
 H3        2.4128200000        4.1148590000        -2.1441810000
FRAGNAME=H2ODFT ! 2
 O1        3.6783070000        3.8351060000        0.8717800000
 H2        3.6020030000        4.1116030000        1.7932670000
 H3        3.4138960000        4.6126330000        0.3648680000
.
.
FRAGNAME=H2ODFT ! 32
 O1        3.7691430000        -1.4091250000        -4.0319510000
 H2        2.8756120000        -1.5562470000        -4.3656710000
 H3        3.6686840000        -1.3995500000        -3.0721400000
 $END
 ####################
```

A.3.2 MD Restart

#########n######### are for restarting MD. For example, if MD stops from errors at t= 39000 fs, a restart geometry and $MD should be obtained from t= 38960 in the run's trj file. #########1######### from trj goes to #########1######### in MD input file and so on with 2 and 3.

```
 .
 .
  #time = 38960 fs
===== MD DATA PACKET =====
NAT=      14 NFRG=      32 NQMMM=       0
TTOTAL=    38960.00 FS    TOT. E=      -180710.543716 KCAL/MOL
POT. E=       -180783.917894 KCAL/MOL  BATHT=           298.000000
KIN. E=            73.374179  TRANS KE=  44.012966  ROT KE=  29.361213 KCAL/MOL
----- QM PARTICLE COORDINATES FOR $DATA GROUP -----
  #########2#########
N           7.0        3.8554852781       -1.5814882196       -3.7440432660
C           6.0        2.9445799802       -1.6872198850       -4.7513152920
 .
 .
H           1.0        3.8827088558       -0.7337182877       -3.1704671239
  ###################
----- EFP PARTICLE COORDINATES FOR $EFRAG GROUP -----
  #########3#########
  $EFRAG
COORD=CART POSITION=OPTIMIZE
FRAGNAME=H2ODFT  !   1
O1                    0.4899097683        5.6815060052        1.3332597175
H2                   -0.1774491436        5.6953737679        2.0005813955
H3
 .
 .
FRAGNAME=H2ODFT  !  32
O1                    1.8397452656       -1.0633683830        1.4432099580
H2                    2.6999380304       -1.1499324935        1.8219625372
H3                    1.3989608369       -0.3812464183        1.9241435987
  $END
  ###################
      GRADIENT DATA (NOT USED BY RESTARTS)...
FRAGMENT #     1  H2ODFT
 .
 .
----- RESTART VELOCITIES FOR $MD GROUP -----

  #########1#########
  $MD READ=.TRUE. MBT=.FALSE. MBR=.FALSE. TTOTAL=    3.90E-11
  MDINT= VVERLET     DT= 0.10E-14 NVTNH= 2  NSTEPS=   11040
  RSTEMP=.T. DTEMP=    25.00 LEVERY= 50000
  RSRAND=.F. NRAND=  1000 NVTOFF=  0 JEVERY=    10
  PROD=.T.   KEVERY=    10 DELR=   0.020
Batht(1)=298.00
  SSBP=.T.   SFORCE=   1.0 DROFF= 12.1*****
TVELQM(1)=     ! QM ATOM TRANS. VELOCITIES (BOHR/PS) !
   -1.436664655E+00  -1.014912632E+00   1.731488079E+01
 .
 .
   -4.714191860E+00  -3.065697154E+00   1.306355299E+01
```

```
TVEL(1)=     ! EFP TRANSLATIONAL VELOCITIES (BOHR/PS) !
  -9.879998843E+00  -1.286351783E+01  -1.625361337E-01
.
.
   4.573922683E+00   9.602851076E+00   6.875495095E+00
QUAT(1)=     ! EFP QUATERNIONS !
   6.658493597E-01  -7.408965451E-01   1.545516153E-02   8.647587874E-02
.
.
   8.096376474E-01   4.753557554E-01   1.423717372E-01   3.134550593E-01
RVEL(1)=     ! EFP ANGULAR VELOCITY (RAD/PS) !
   1.392338986E+01  -1.145718732E+01  -1.039863492E+01
.
.
  -1.217796312E+01   2.229922184E+00   2.270460047E+01
QUAT1D(1)=     ! EFP QUATERNION 1ST DERIV. !
   4.237107071E+12   4.149735474E+12   8.424472510E+12   1.422917746E+12
.
.
   5.909828315E+12  -1.125106858E+13  -8.408659179E+11   2.179439222E+12
QUAT2D(1)=     ! EFP QUATERNION 2ND DERIV. !
  -6.427981808E+26  -4.203604186E+26  -7.365308723E+25   1.102236012E+26
.
.
   8.379527677E+26   4.204660564E+26  -1.448608909E+27  -2.676740098E+27
 $END
 ####################

 #time = 38970 fs
===== MD DATA PACKET =====
NAT=      14 NFRG=      32 NQMMM=       0
TTOTAL=    38970.00 FS    TOT. E=      -180715.284973 KCAL/MOL
POT. E=       -180783.048242 KCAL/MOL  BATHT=           298.000000
KIN. E=            67.763269  TRANS KE=  41.754223  ROT KE=  26.009046 KCAL/MOL
----- QM PARTICLE COORDINATES FOR $DATA GROUP -----
N           7.0        3.8518322659       -1.6019379232       -3.6477010146
.
.
```

A.3.3 TD-DFT Input File

Excited state energies are calculated using TD-DFT. Direct SCF calculation is turned on. Basis set = 6-311++G(2d,p)

```
 # run type = [excitation] energy, with functional = CAMB3LYP, and TDDFT
 $CONTRL SCFTYP=RHF TDDFT=EXCITE DFTTYP=CAMB3LYP RUNTYP=ENERGY
       ICHARG=0 MULT=1 COORD=UNIQUE MAXIT=200 $END
 #TDDFT requires lots of memory space
 # memory requested at each node =1000 million words
 # memory reserved for communication = 1000 million words
 $SYSTEM MWORDS=200 MEMDDI=250 $END
 #activate direct SCF calculation
 $SCF DIRSCF=.T. $END
 # find 5 excited states - the current setting is purely driven by its lower cost
 # Previous experience shows that 10 states gives only a few strong peak.
  ↪
 $TDDFT NSTATE=5 TPA=.f. $END
```

```
 # Basis set = 6-311++(2d,p)
 $BASIS GBASIS=N311 NGAUSS=6 NDFUNC=2 NPFUNC=1
       DIFFSP=.TRUE. DIFFS=.TRUE. POLAR=POPN311 $END
 # solute geometry - C1 1 = symmetry data
 $DATA
aniline32 at t= 15010
C1 1
N        7.0        2.4008547653        5.9114221893        -1.1412310058
C        6.0        1.9371475177        5.9223533811        -2.4157851823
C        6.0        0.6209366009        6.2361805033        -2.7812041720
.
.
H        1.0        1.7323956480        5.6459040114        -0.4329519914
H        1.0        3.3564486514        5.6102990678        -1.0242941608
 $END

 # solvent geometry in EFP1 (EFP2 is still not available)
 $EFRAG
COORD=CART  POSITION=OPTIMIZE
FRAGNAME=H2ODFT ! 1
O1         2.335993939511        3.751856628604        1.427418842826
H2         1.439322965739        3.833168541986        1.710699607266
H3         2.854351938959        3.613582975500        2.203990690907
FRAGNAME=H2ODFT ! 2
O1         3.266753260182        2.613267395174        3.808546039622
H2         3.514552920456        1.768271678250        3.468757839439
H3         3.822697636667        2.780022240614        4.552855871622
.
.
FRAGNAME=H2ODFT ! 32
O1        -0.041331901955       -3.906598195206        1.282021099515
H2        -0.790424236756       -4.406259423929        1.565001283174
H3        -0.374908444181       -3.111491773717        0.898079798177
 $END
```