

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

Kasidet Trerayapiwat* and Soren Eustis*

Department of Chemistry, Bowdoin College, Brunswick, ME

E-mail: ktteraya@bowdoin.edu; seustis@bowdoin.edu

Abstract

Understanding of the changes in molecular electronic structure following the absorption of light is a fundamental challenge for the goal of predicting photochemical rates and mechanisms. Excitation energies and oscillator strengths were calculated using different theories and methods. A new approach was selected to model the photon absorption: Molecular Dynamics Time Dependent Density Functional Theory (MD-TD-DFT). An aniline molecule equilibrates in the presence of a number of water molecules at room temperature using 6-311++G** basis set. Excitation energy and oscillator strength of aniline geometries in equilibrium are then calculated using TD-DFT with CAMB3LYP, or M06-2X functional. The computed physical properties from MD-TD-DFT then compared with data from experimental absorption spectra to evaluate the accuracy of the methods. Absorption spectras underlying modified Gaussian functions were decomposed and integrated to calculate experimental oscillator strength at a certain excitation energy using an R code written by Peter Cohen.

Introduction

Environmental Photochemistry And Micropollutants

During the past century, more and more synthetic chemicals has been created for commercial pharmaceuticals and personal care products. Each year, about 300 million tons of organic chemicals are being added into water systems.¹ These chemicals has been detected in ng/L to g/L in the aquatic systems worldwide.² The scale of chemicals being introduced into the water reaffirms the need for close studies in order to understand the consequences of these compounds on the environmental system, and eventually to living beings. While major toxic chemicals, such as CCl₄, DDT etc., has been constantly regulated or banned by governments for its effects on agriculture and the environment, many more compounds have not received the attention they deserve. Even though some are approved as safe for daily usage in household products, their impacts after being disposed into water remain largely unknown due to

exhaustive amount of effort to study all the compounds. These low-concentration chemicals are collectively called micropollutants. However, small concentration, large number of different types, difficulty separating or concentrating for analysis contribute retard the effort to understand their effects on in water systems. Nevertheless, in rivers close discharge areas, the observed attenuation in concentration of micropollutants from upstream to downstream has been identified to be mainly from photolysis.³⁻⁶ Some photo-degrade into more toxic species. Triclosan, which been used as an anti-bacterial agent in household soap and health care products decomposes under sunlight to Dioxins and PCBs, well-known carcinogens.⁷ Previously, computational studies of triclosan in the excited states were carried out by Soren N. Eustis.⁸ Within lab, Nathan Rieke '14 worked on applying computational model to estimate quantum yields of photo-excited small organic molecules.⁹

Computational approach, Photoexcitation, and UV-VIS Spectrum

In the recent years, computer has become more and more accessible to researchers.¹⁰ The power of High Performance Computer (HPC) expands the universe of rigorous computational calculations to analytically unsolvable quantum problems. Moving away from semi-empirical models, computational chemists can now implement Ab Initio methods using fundamental equations, which decades before, was prohibited by the high computational time and resources. In studying photo-degradation in micropollutants, computational approach allows for a priori prediction of photo-products and their impact on the environment instead of posteriori study of damage done, not to mention the complexity and limitations of experimental designs in order to study individual species of micropollutants. In natural water, many species of micropollutants interfere with each other putting limits on how much experimental approach can be used to study the natural system.

A molecule in its ground state can absorb a photon, transform to its first singlet excited state (S1), then its first triplet excited state (T1) as illustrated in figure 1. Triclosan at triplet states goes through photo-degradation and decomposition. Understanding transition

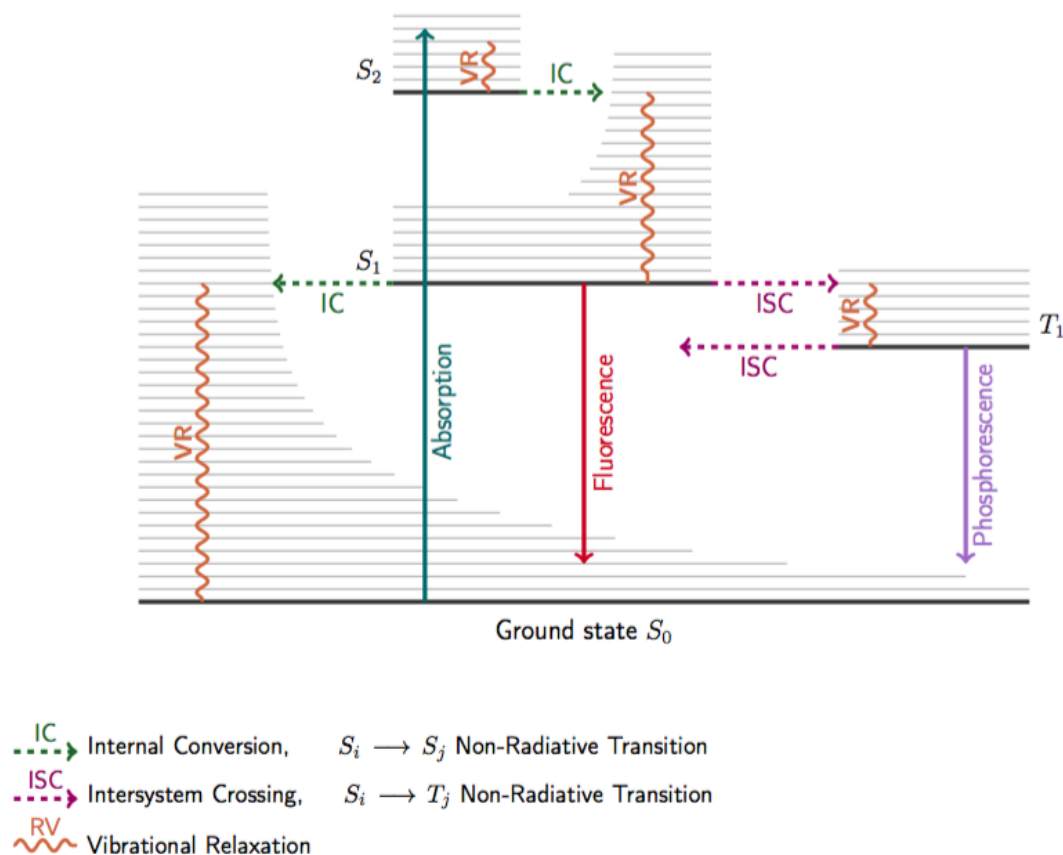


Figure 1: Jablonski diagram. A possible pathway of photo-excitation starts at ground state to singlet excited state, then triplet excited states.

from singlet to triplet state was studied previously in Eustis lab.⁹ While reactions in excited state are important to understanding photochemical reaction, excitation from ground states by photons to excited states is equally important to understand complete reaction mechanism. Most familiarly, studying absorption of photons in chemical compounds are done through UV-VIS absorption spectra. A molar absorptivity vs wavenumber plot reveals excitation energies of electronic transitions in a particular species of chemical compound. In gas phase, simulation of UV-VIS spectrum can be done more easily due to lack of solvent broadening.¹¹ However, in linking UV-VIS spectrum with theory, a more fundamental value called oscillator strength should be used. 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 -

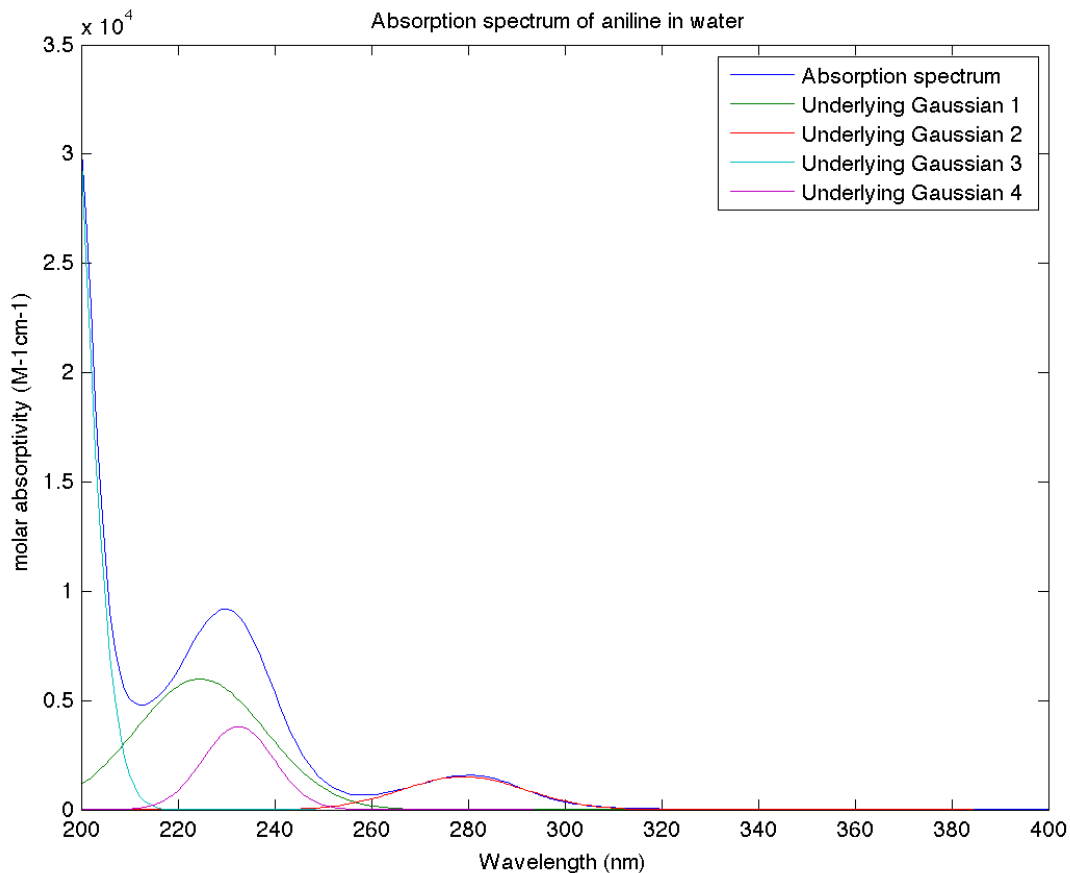


Figure 2: Absorption spectrum of aniline in water with underlying Gaussian printed out. Program Fityk is used to fit these four Gaussian manually.

see equation 1.¹²

$$f = 4.39 \times 10^{-9} \int_{\tilde{\nu}_a}^{\tilde{\nu}_b} \varepsilon(\tilde{\nu}) d\tilde{\nu} \quad (1)$$

After calculation of excited state energies and the oscillator strength, computational results will be compared with experimental UV-VIS spectrum to evaluate accuracy of the models used. One another complication of the problem is the solvent broadening of absorbance peaks. As peaks are broadened, each individual peak combines together into a continuous experimental spectrum as in figure 2. Integration cannot be done without deconvoluting the underlying Gaussians. Currently, Matlab code utilizing Bayesian probability as a statistical technique to properly find the Gaussian plots are developed by professor Soren Eustis from

previous work in R with Peter Cohen '18.

There are currently no studies using full statistical method on quantitative calculation of excited state energies of organic molecule in water. This allows room for a systematic approach to develop an appropriate computational model that would allow for further understanding of aquatic pollutants.

Solvent Models

Despite recent advent of growth in computer speed and burgeoning interest in incorporating computational models to further understand the nature world, large systems such as solvation models remains a big challenge.¹³ In modeling effects of solvent molecules on solute, implicit solvation models were previously implemented because it allows for acceptable results calculation while maintaining good speed (low computational cost). Most famous of all implicit models is Polarizable Continuum Model (PCM).¹⁴ 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. Its downfall is that, however, its accuracy falls short of static and dynamic contribution of excited states properties.¹⁵ Furthermore, implicit solvent model also neglects hydrogen-bonding as it assumes implicit implementation in dispersion forces and electrostatics.¹⁶ Especially in calculating excited state energies, an accurate solvent model should be used.¹⁷ In explicit solvent model, one recent notable method Effective Fragment Potentials (EFP) can be used to model explicit solvents with non-bonded van der Waals interactions, hydrogen bonding using Coulomb interactions, polarization, and exchange repulsion without high computational expense of explicit models.^{18,19}

Computational Models: Theories, Basis Sets, and Functionals

Among all current theories, Time-Dependent Density Functional Theory (TD-DFT) is the most promising with its high accuracy when used with appropriate functionals and low

computational cost.²⁰ Implementing EFP solvent model, TD-DFT can be used to accurately calculate excited state energy of acetone in water.¹⁹ Typically in Implicit solvent model, geometry optimization of solute molecule is carried out with PCM, followed by calculation of excited state energies, also with PCM. This static ground state molecule however does not accurately represent solute in water.²¹ Instead, Molecular Dynamics (MD) of solute and solvent fragments can be used to obtain a range of equilibrated structures for excited state energies calculation. Mark Gordon et al. averaged the calculated energies of each excited state to arrive at a final excited states energy.²¹

According to previous basis set studies, while having roughly the same computational cost, an average-sized basis set 6-311++G(2d,p) (ref needed) performs better than aug-cc-pVDZ (ACCD) (ref needed).^{22,23} 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} . In previous study, CAMB3LYP (ref needed), M06-2X (ref needed) are reported to be one of the most reliable functionals in calculating vertical excitation energy (average root mean square deviation of 2272 cm^{-1} and 2187 cm^{-1} respectively compared to 3114 cm^{-1} for M06 (ref needed) and 18196 cm^{-1} for CIS) (ref needed).²³

Solvent Boundary Potential

According previous computational results, solvent boundary condition (ssbp) is required to model solvent effect outside of explicit model. Especially for the equilibrium, solvent molecule can evaporate off the water cluster. Figure ?? shows two evaporated water molecule at 1500 fs. During the evaporation, the potential energy of the system increases momentarily, before slowly decreases as the system equilibrated. A method of using potential boundary is proposed to simulate force field around target molecule. When a solvent fragment crosses an empirically-assigned radius, potential wall will exert slight force to push the molecule back toward the center.²⁴ As noted in the paper, ssbp radius is set empirically from trial and

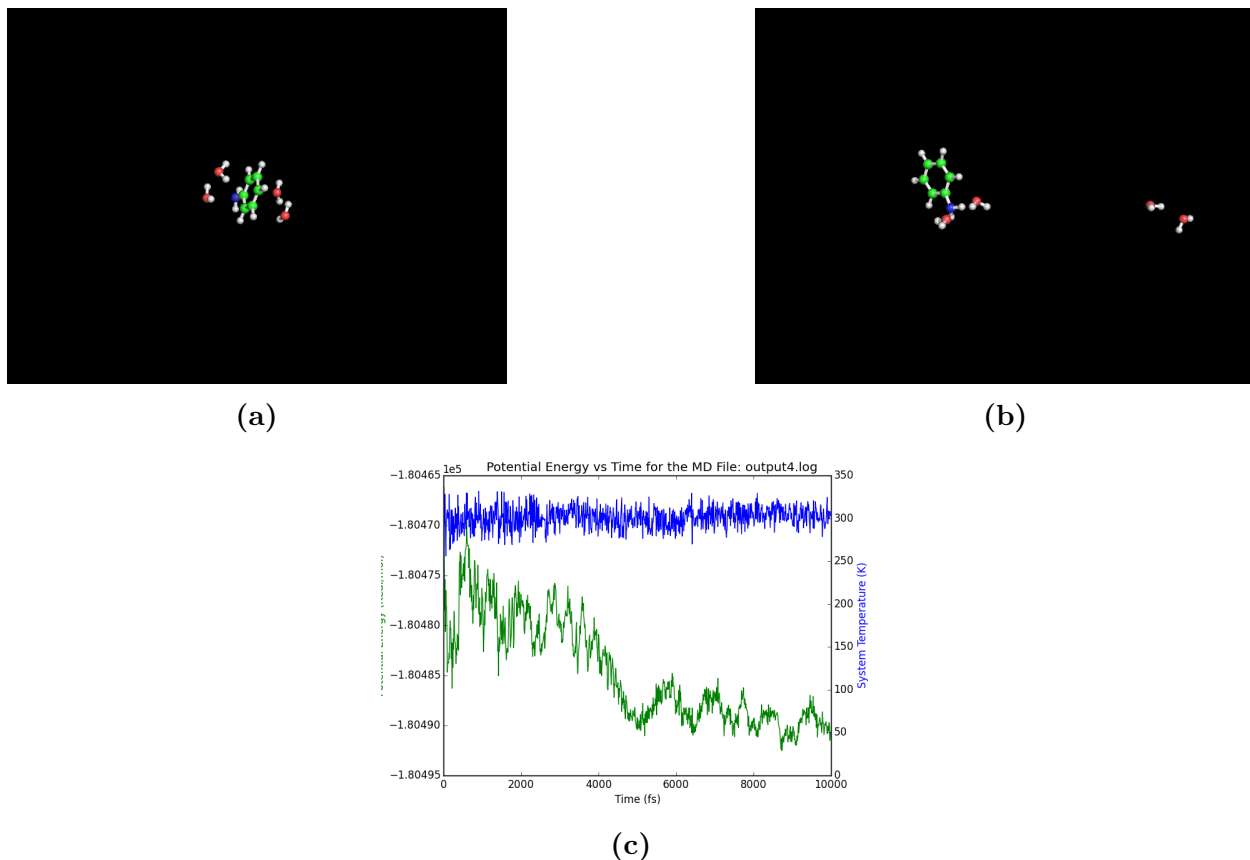


Figure 3: Molecular dynamics of aniline with 4 water without solvent boundary condition. Notice a) Initial aniline and water geometry b) System after 1500 fs. c) the potential energy of the system

error (see Appendix for prepareMD2.py for estimating ssbp radius). Also in GAMESS (ref needed), 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 potential boundary is $V = 0.5 \times SFORCE \times (ssbp\ radius)^2$.

Method

Overview

Systematic approach to developing computational model illustrated in figure 4. Python scripts are used to facilitate some of the steps (see Appendix). Starting molecule geometries can be obtain from optimizing molecular geometry of solute and solvent molecules yielding `solute.xyz` and `solvent.xyz`. These two files are then inputed into Packmol program (ref needed) with empirical starting outer radius and number of water molecules to create starting geometry of solvated solute. After equilibrium has been reached, a 10 ps of equilibrated geometries are used in calculating excited state energies using TD-DFT in GAMESS. After comparison among all equilibrated systems, number of water will be decided before moving on to other functionals. Once a functional is decided, the model can be applied to other molecules.

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. R was used to fit Gaussian under absorption spectrum, and integrate to find oscillator strength. In order to most accurately calculate the excited energies, the 6-311++G(2d,p) basis set is chosen to run single point energies on a set of equilibrated geometries. It has been decided to reduce the basis set in running MD, a smaller basis set 6-31+G(2d,p) (ref needed) will be used in order to cut computational cost. The decision comes as a suggestion from professor Soren Eustis after weeks of waiting for computational results when determining the number of water molecule in the model. Two best-performing DFT functionals out of all examined in previous study are explored: CAMB3LYP, M06-2X.²³ Either PBE0 or HSE06 will also be used to in future work.

The EFP model of water molecules is chosen to implement an explicit solvent model to

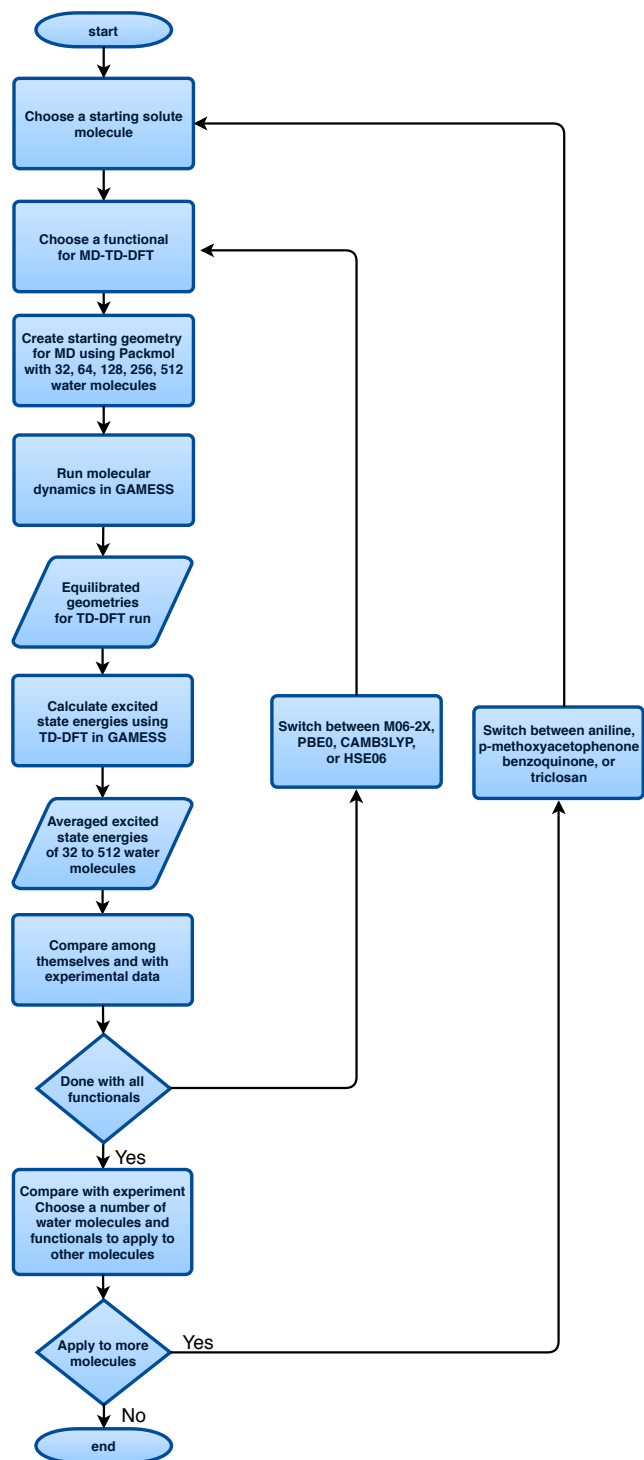


Figure 4: Flowchart of work in this project. Some of these processes are done using python scripts

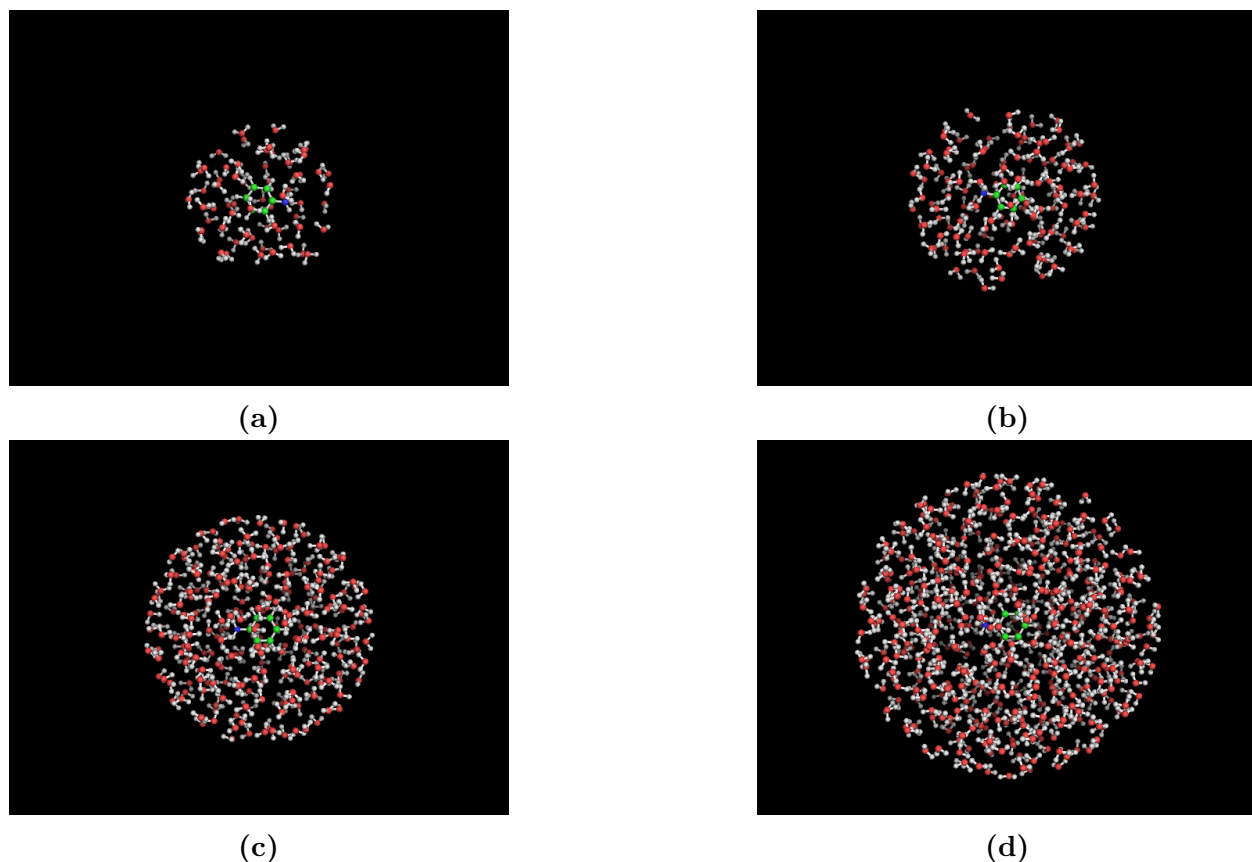


Figure 5: Binary approach to determining the right number of water in solvation sphere. Five numbers of water molecules between using 2^n with $n=5-9$ are used. a) 64 water molecules b) 128 water molecules c) 256 water molecules d) 512 water molecules

calculate excited states energies. The appropriate number of water molecules to be included as EFP in the model has never been evaluated. Too many waters results expensive computational cost. Too few water will not fully model the solvent. A binary framework will be used to test how many water molecules are needed to fully solvate the solute environment: 32, 64,...512. Once excited state energies for each system are calculated, the results will be compared with experimental spectral data to find optimum number of waters before determining which functional should be chosen to achieve the most accurate computational model.

Aniline has been chosen as our initial test solute due to its hydrogen bonding capability, its small size, and extensive studies on aniline within Eustis lab. Successful model on aniline is likely to be applicable even to larger molecule with similar properties. After a model for

aniline is decided, para-methoxyacetophenone and benzoquinone will be studied for their similarity in functionals to triclosan. Finally, the model will be applied to more known aquatic pollutants such as triclosan.

Preliminary Results and Discussions

Only aniline with 32 water with CAMB3LYP has excited state energy tabulated. Time was spent trying to fix bugs in GAMESS on Bowdoin HPC and ssbp problem. However, once everything is fixed, the process should run relatively smoothly for other functionals. Experimental absorption spectrum of aniline in water was collected by Alex and Holly over the summer of 2015.

Python Scripts

Much of time was also spent on writing python scripts (see Appendix). In order to automatically generate input files and cultivate output data from output files, many python scripts are written from scratch. Since scripts are specific to each GAMESS run, there is a limited number of scripts available on the internet (virtually none for this project). Log files obtained from GAMESS contains both valuable experimental data and useless text strings. Python scripts play an important role in both data collection and smoothing up the process between each computational steps. For example, even though WEBMO (ref needed) can generate sets of latest geometry in MD run, but retrieving geometry from each MD step requires one to manually open the log file and copy-paste the geometry into input files of the next step one by one. The python script postMDDDataPull2.py is designed to pull thousands of geometries and generate GAMESS input files for TD-DFT energy calculation within seconds. Generating these python scripts will also allow unified program to be developed in order to automate the whole project without any manual input.

Determining the equilibrium

Determination of equilibrium was determined by eyeballing a plot of the solvent solute system's potential energy over time for a stable period as shown in the figure 6d. The consistently low fluctuation indicates the start of equilibrium at 15010 fs. 1000 frames or

10000 fs of MD geometries were used to calculate the excitation energies in TD-DFT run. Linear regression fit results in $2.18 \times 10^{-4} \text{ kcal} \cdot \text{mol}^{-1} \text{ fs}^{-1}$. 2.18×10^{-4} will be used as a threshold for future calculation. Slope of linear fit for each 10 ps interval starting at time on x-axis is displayed on figure 7. Fortunately, the fitted slopes do cross over 0 meaning the system do reach equilibrium, not approaching it.

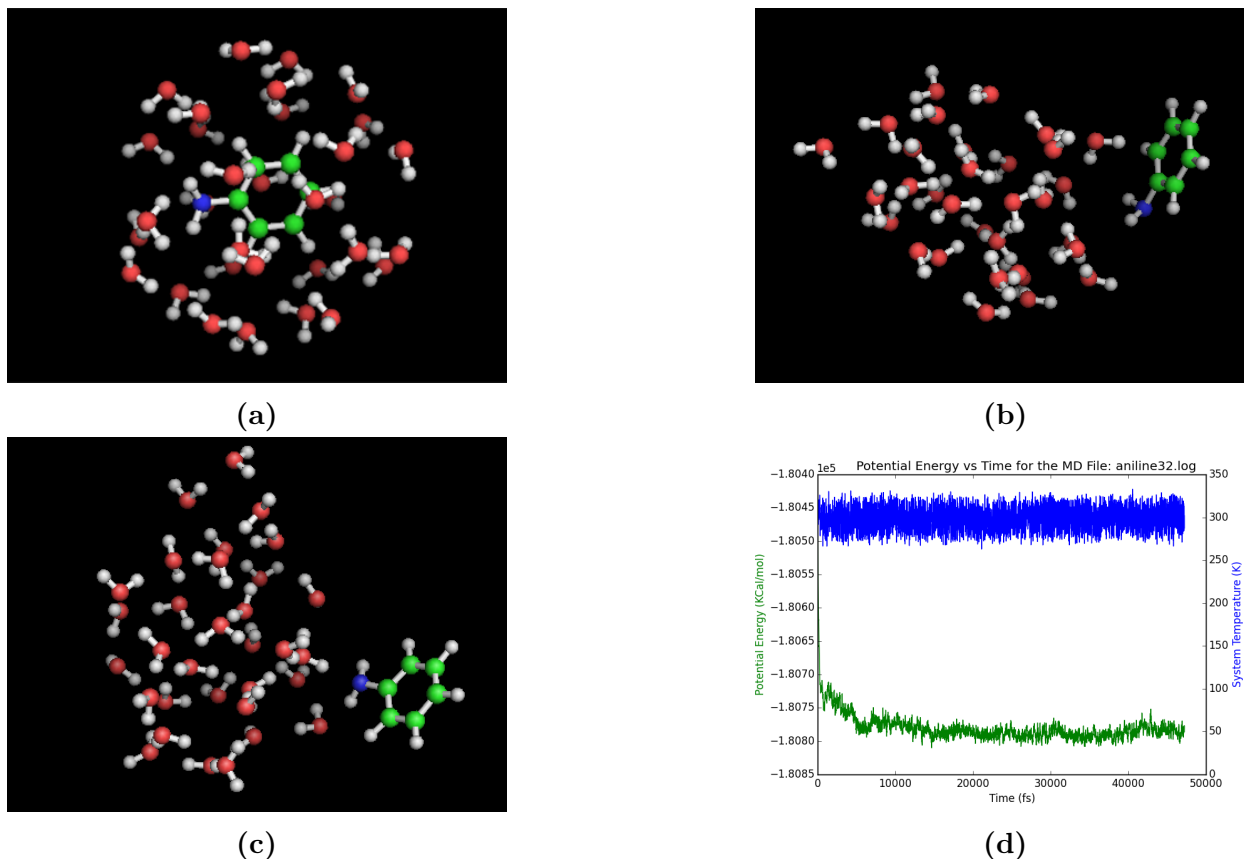


Figure 6: Molecular Dynamics run of aniline in 32 explicit solvating water molecules. Notice that at equilibrium, aniline molecule comes outside of the water sphere. Albeit hydrogen bond being clearly established, lack of total submersion in water means 32-water does not fully solvate the aniline molecule and suggests that 64-water will give more accurate results. (a) starting geometry of MD run created by Packmol. (b) geometry after 15000 fs. Notice the hydrogen bond between the amino group and water cluster. (c) geometry after 25000 fs. The amino group is pointing in the water sphere, as it continues to through out the whole MD run. (d) A plot of potential Energy of the system vs time. At 15000 fs, equilibrium starts as evident by decrease in energy fluctuation.

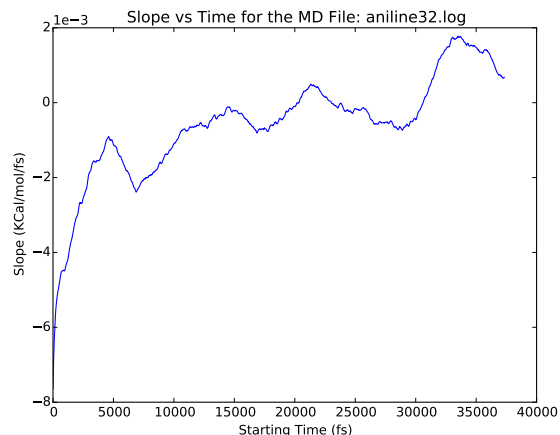


Figure 7: Plot of linear fitted slope of aniline with 32 waters vs time. The slope almost reaches 0 at 15010 to 25000 fs. There are slope value that reaches $2 \times 10^{-7} \text{ kcal} \cdot \text{mol}^{-1} \text{ fs}^{-1}$ at 20200 fs to 30100 fs.

Determining the optimum solvent environment

TD-DFT calculation for aniline with 32, 64, 128, 256, 512 surrounding water molecules were performed with CAMB3LYP basis set. Firstly, for 32 water molecules, the equilibrium were chosen to start from 15 ps and the stopping point of calculation was 25 ps; 1000 jobs for every 10 fs. Geometry of the system though challenge the accuracy of 32-water model. Aniline molecule surrounded in 32 water molecules is unfortunately most stable not being fully solvated. Aniline can be seen outside of the water cluster at the time of equilibrium. This is in contrast to expected 32 water as the first solvation shell for aniline.²⁵

Table 1: Wavelength and Oscillator Strength from MD-TD-DFT calculation of aniline in 32 water molecules.

Wavelength (nm)	Oscillator Strength
173.00	0.165685
180.20	0.364739
184.99	0.339029
214.30	0.143915
246.26	0.0383513

Excited state energy and oscillator strength in each excited state at different equilibrated

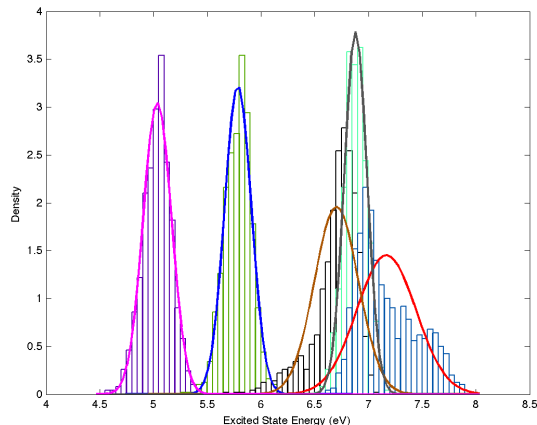


Figure 8: Histogram plot of excited state energies calculated from equilibrated geometries in eV. Peaks of each Gaussian plot represents the mean values of excited state energies as tabulated in table 1 in wavelength unit

geometries can be fitted to a normal distribution as in figure 8. This approach results in average excited state energies and oscillator strengths. The excited state energies and its oscillator strength from MD-TD-DFT calculation of aniline with 32 waters are tabulated in table 1. When compared with aniline’s UV-VIS spectra, as in figure 2, there are several problems. Firstly, the calculated value at 246 nm does not accurately capture the peak at 230 nm and there is no calculated excitation energy at 280 nm, where the experimental peak is. The problem is probably due to aniline not being fully solvated. Wavelengths and oscillator strengths from experimental spectral data calculated by Fityk are tabulated in table 2. Two calculated wavelengths, 214.30 nm and 246.26 nm, are identified with the experimental peak observed at 225 nm and 279 nm with comparable oscillator strength value. Even though

Table 2: Wavelength and oscillator strength calculated from experimental UV-VIS spectrum using Fityk, figure 2. Notice the first peak does not have a maximum, this is due to the lower limit of the spectral data.

Wavelength (nm)	Oscillator Strength
<200	>0.1298
225	0.1699
232	0.0570
279	0.0274

MD calculations for CAMB3LYP aniline with 64 through 512 waters have completed, no TD-DFT energies has been calculated as of now.

Next steps

Once energies and oscillator strengths of each solvent models are computed, data as a function of number of water can be evaluated. Aniline with 512 waters is the most time consuming system. If there is no significant improvement in comparison with 256 waters, this model will be discarded. In the future R code will be used instead of Fityk since Fityk requires manual insertion of underlying Gaussian plots. Elimination of this process will results in less manual input and more statistical approach to a statistical problem. Calculation using other functionals can be done relatively faster once the optimum number of water framework has been establish.

Appendix

Python Scripts

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 #####
4
5 import sys
6 import csv
7 import os
8 import string
9
10
11 #for asking what the input in terminal should be
12 try:
13     if str(sys.argv[1])=='?':
14         print '\nCall function as: prepareMD.py input.xyz
15             ↪ numberOfSoluteAtoms numberOfSolventAtoms
16             ↪ numberOfSolventMolecules \n'
17         sys.exit()
18 except IndexError:
19     print '\n!!!Input command Error. Call function as: prepareMD.py input.xyz
20         ↪ numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules \n'
21     sys.exit()
22 #for assigning received input from terminal
```

```

20  try:
21      input=str(sys.argv[1])
22      numberOfSoluteAtoms=int(sys.argv[2])
23      numberOfSolventAtoms=int(sys.argv[3])
24      numberOfSolventMolecules=int(sys.argv[4])
25  except IndexError:
26      print '\n!!!Input command Error. Call function as: prepareMD.py input.xyz
    ↪      numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules \n'
27      sys.exit()
28  #generate output name
29  if input.endswith('.xyz'):
30      output = input[:-4]+'.inp'
31  #for safety - at worst the output will not overwrite the input
32  else:
33      output=input+'.inp'
34
35  #part one
36  #This part is for finding ssbp radius for inout file
37  #enumerate gets data in line - line and line index - n
38  radiusInSolute=0.0
39  radiusInSolvent=0.0
40  avgX=0.0
41  avgY=0.0
42  avgZ=0.0
43  X=[]
44  Y=[]
45  Z=[]
46
47  lineNumber=0
48  #open input
49  f2=open(input)
50  for line in f2:
51      lineNumber+=1
52      #first two line does not contain useful info - x y z start on the third
    ↪      line
53      if lineNumber>2:
54          #x y z
55          lineSplit=line.split()
56          X.append(float(lineSplit[1]))
57          Y.append(float(lineSplit[2]))
58          Z.append(float(lineSplit[3]))
59  #for looping through array below
60  size=len(X)
61  #find a CG for solute atoms
62  avgX=sum(X[:numberOfSoluteAtoms-1])/numberOfSoluteAtoms
63  avgY=sum(Y[:numberOfSoluteAtoms-1])/numberOfSoluteAtoms
64  avgZ=sum(Z[:numberOfSoluteAtoms-1])/numberOfSoluteAtoms

```

```

65  #looping to find radius of each atoms in relative to solute's CG
66  #also find the maximum value of them
67  for i in range(0,size):
68      d=((X[i]-avgX)**2+(Y[i]-avgY)**2+(Z[i]-avgZ)**2)**0.5
69      if i<numberOfSoluteAtoms:
70          if radiusInSolute<d:
71              radiusInSolute=d
72      else:
73          if radiusInSolvent<d:
74              radiusInSolvent=d
75
76  #radius should be a little bit larger than the two combined - 3 Angstrom larger -
77  → this does not need to be super accurate
78  radiusInSolute=radiusInSolute
79  radiusInSolvent=radiusInSolvent
80  ssbpRadius=radiusInSolute+radiusInSolvent+3
81  print '\n'
82  print 'Radius in solute is:\t'+str(radiusInSolute)
83  print 'Radius in solvent is:\t'+str(radiusInSolvent)
84  print 'ssbp Radius should be:\t'+str(ssbpRadius)
85  print '\n'
86  #####
87  #Part two - this is where geometry data is taken from xyz, change into GAMESS
88  → input's format + other input
89  numberOfAllSolventsAtoms=numberOfSoventAtoms*numberOfSolventMolecules
90  fragmentNumber=1;
91  atomLabel=1
92  #this dict is for generating atomic number from Acronym

```

```

93 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}

94
95 #write out put the headers - all the commands for GAMESS + ssbp
96 #functional = M06-2X - DFTTYP=M06-2X
97 f = open(output, 'w');
98 f.write('' $CONTRL SCFTYP=RHF RUNTYP=MD COORD=UNIQUE
99     DFTTYP=M06-2X MAXIT=200 ICHARG=0 MULT=1 $END
100     $MD KEVERY=10 PROD=.T. NVTNH=2 MBT=.T. MBR=.T.
101     BATHT=298 RSTEMP=.T. DTEMP=25 NSTEPS=50000
102     SSBP=.T. SFORCE=1.0 DROFF='' +str(ssbpRadius)+'' $END
103     $DFT DC=.F. $END
104     $SYSTEM MWORDS=1000 MEMDDI=1000 $END
105     $SCF DIRSCF=.T. $END
106     $BASIS GBASIS=N31 NGAUSS=6 NDFUNC=2 NPFUNC=1
107     DIFFS=.TRUE. POLAR=POPN311 $END
108     $DATA\n'' + 'MD INPUT for' +input+'\nC1 1\n'')
109
110 #geometry
111 with open(input) as f1:
112     #read by line
113     #readlines if okay to use bc xyz is not too big
114     lines = f1.readlines()
115     #enumerate gets data in line - line and line index - n
116     for n, line in enumerate(lines):
117         #take all solute molecules (in range of 2 (line 3 where packmol starts)
118         ↪ to num+2)
119         #it's num+2 bc the range will go to num+1
120         if n == 2:

```

```

120     print 'Now Writing Solute:\n'
121 if n in range(2,numberofSoluteAtoms+2):
122     lineSplit=line.split();
123     lineSplit.insert(1,str(atomicNumber[lineSplit[0]]))
124     #convert coordinates to 10 decimals (add zeros if need be)
125     for index in [2,3,4]:
126         lineSplit[index]=float(lineSplit[index])
127         lineSplit[index]=format(lineSplit[index],'.10f')
128         grandString=lineSplit[0]+'\\t'+lineSplit[1]+'\\t'+lineSplit[2] +
            ↪ '\\t'+lineSplit[3]+'\\t'+lineSplit[4]+'\\n';
129     f.write(grandString)
130     print grandString
131 if n == numberofSoluteAtoms+2:
132     f.write(' $END\\n\\n $EFRAG\\nCOORD=CART POSITION=OPTIMIZE\\n')
133     print 'Now Writing Solvent:\n'
134     #now start doing solvent - (need to add fragment number and atom labels)
135     startPointOfSolvent=numberofSoluteAtoms+2
136     if n in range(startPointOfSolvent,
            ↪ startPointOfSolvent+numberOfAllSolventsAtoms+1):
137         #atomlabel = O1, H2, H3 from O, H, H
138         if atomLabel%numberOfSoventAtoms==1:
139             grandString='FRAGNAME=H2ODFT ! '+str(fragmentNumber)+'\\n'
140             f.write(grandString)
141             print grandString
142             fragmentNumber+=1;
143             atomLabel%=numberOfSoventAtoms
144             lineSplit=line.split();
145             lineSplit.insert(1,str(atomLabel))
146             atomLabel+=1
147             #convert coordinates to 10 decimals (add zeros if need be)
148             for index in [2,3,4]:
149                 lineSplit[index]=float(lineSplit[index])
150                 lineSplit[index]=format(lineSplit[index],'.10f')
151             grandString=' '+lineSplit[0]+lineSplit[1]+'\\t'+lineSplit[2] +
                ↪ '\\t'+lineSplit[3]+'\\t'+lineSplit[4]+'\\n';
152             f.write(grandString)
153             print grandString
154     #close the inp with $END
155     f.write(' $END\\n')
156     #####

```

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 femtosecond - in the current MD input file - see MD Input File section).

```

1 #####
2 ### 3dExtract pulls out geometries from MD run and make ###
3 ### an xyz-movie file for inspection MD          progress          ###
4 #####
5
6 import os as os
7 import sys
8
9 #for asking what the input in terminal should be
10 try:
11     if str(sys.argv[1])=='?':
12         print '\nCall function as: 3dExtract.py input.log
13             ↪ numberOfSoluteAtoms numberOfSolventAtoms
14             ↪ numberOfSolventMolecules \n'
15         sys.exit()
16 except IndexError:
17     print '\n!!!Input command Error. Call function as: 3dExtract.py input.log
18         ↪ numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules \n'
19     sys.exit()
20
21 #Call as 3dExtract.py inputfile #ofsoluteAtom #ofsolventAtom #ofsoluteMolecules
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: 3dExtract.py input.log
29         ↪ numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules \n'
30     sys.exit()
31
32 if input.endswith('.log'):
33     output = str(input[:-4])+'.xyz'
34 else:
35     output=str(input)
36
37 numberOfAllSolventsAtoms=numberOfSoventAtoms*numberOfSolventMolecules
38 #This is for comparing files to be written
39 previousGrandString=''
40 collectionStarted=False
41 time=''
42 #1 is for cartesian line (useless), then 1 in 3(n+1) is for fragment H2O line
43     ↪ (also useless)
44 numberOfLinesToBecollected=numberOfSoluteAtoms+1+numberOfSolventMolecules*(numberOfSoventAtoms

```

```

39
40
41 #number of molecules so far
42 timeCount=0
43 #total number of atoms (solute + solvent) - used later in checking if file is
44   ↪ complete
45 atomCount=0
46 #define functions here
47 lineSinceTimeIsFound=0;
48 #do an input of solvent, solute atoms
49 mollist=[]
50 #for printing time
51 def printTime (thisLine):
52     lineComponents=thisLine.split();
53     timeString=str(lineComponents[3]);
54     print "Analyzing t = "+timeString+" fsec\n"
55 #to determine if line should be collected -
56 def shouldCollect():
57     #only check if collection is in progress - if it is, then continue to finish
58     ↪ collecting the lines
59     #collectionStarted is determined when 'QM ATOM COORDINATES (ANG)' is found
60     if collectionStarted:
61         #from first solute atom to the last fragment atom
62         if (atomCount>=0 and atomCount<numberOfLinesToBeCollected):
63             return True;
64         else:
65             return False;
66     else:
67         return False;
68 # only write when atomCount==numberOfLinesToBeCollected
69 def shouldWrite():
70     #only check if collection is in progress
71     if collectionStarted:
72         #solute
73         if (atomCount==numberOfLinesToBeCollected):
74             return True;
75         else:
76             return False;
77     else:
78         return False;
79 #Even now I still don't understand why GAMESS duplicate system geometry for a
80   ↪ step twice in the log file
81 #This is written to prevent duplication of geometry in the xyz-movie file
82 def moleculeIsNotADuplication(currentMoleculeToBeWritten):
83     # to compare previously stored geometry and a new one is tricky bc each
84     ↪ string has different lengths

```



```

82         # there must be a better of doing this - note for possible place for
           ↪ improvement
83         #current the speed is quite slow probably due to this step
84         halfSize=int(len(previousGrandString)/2)
85         threeQuartersSize=int(len(previousGrandString)*3/4)
86         if previousGrandString=='':
87             return True;
88         if (currentMoleculeToBeWritten[halfSize:threeQuartersSize] not in
           ↪ previousGrandString[halfSize-1:threeQuartersSize+1]):
89             return True;
90         else:
91             return False;
92
93     #clear output.xyz
94     f = open(output, 'w');
95     f.write('')
96     #open input
97     f1=open(input)
98     #enumerate gets data in line - line and line index - n
99     #readlines() is eliminated because it creates a huge array and python cannot
       ↪ handle it when log file get very large
100    #using for line in... alleviate the burden on memory and actually speed up the
       ↪ process
101    for line in f1:
102        #this keyword is usually before coordinate
103        grandString=''
104        #find out if checking for collectionStarted is needed
105        if shouldCollect():
106            #split line
107            lineSplit=line.split()
108            atomCount+=1;
109            #append to molList
110            molList.append(lineSplit)
111            # if this then start collecting
112            elif 'QM ATOM COORDINATES (ANG)' in line:
113                collectionStarted=True
114            #lastly, if none of the above, then find and print time
115            elif '*** AT T=' in line:
116                time=str(line)
117                printTime(line);
118
119        if (shouldWrite()):
120            atomCount=atomCount-(numberOfSolventMolecules+1);
121            #for loop through a ***COPY*** of molList and delete some element from
           ↪ molList!
122            #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

```

```

123     for line in list(molList):
124         #if line has 4 elements then it's a coordinate from solvent fragment
125         ↪ - we have to drop number behind atom - 01 to 0
126         if len(line) == 4:
127             #store string
128             oldString = line[0]
129             #replacement string
130             newString=''
131             #loop to check if it's a alphabet or not
132             for character in range(len(oldString)):
133                 #do substring of 1 character
134                 subString = oldString[character:character+1]
135                 #check if it's an alphabet - yes? then add to newString
136                 if subString.isalpha():
137                     newString = newString + subString
138                 #replace '01' with '0'
139                 line[0]=newString
140             #if it's 5 then it's solute coordinate - we have to get rid of atomic
141             ↪ number behind atomic representation
142             elif len(line) == 5:
143                 # 'N 7.0 ...' will become 'N ...'
144                 del line[1]
145             #the rest are crap - just remove it out of the line
146             else:
147                 #there's a reason why this is remove - not del - since we are
148                 ↪ iterating if we delete using index we are gonna be screwed
149                 molList.remove(line)
150             #this is for if we have an incomplete file or inconsistant number of
151             ↪ atoms we should only use the one before and break for loop without
152             ↪ appending to grandString
153             if len(molList) != atomCount:
154                 print 'error'
155             #xyz file has a format that we need atomCount at the top followed by
156             ↪ snapshot number(timeCount) on the next line before adding any
157             ↪ coordinates
158             grandString=grandString+str(atomCount)+'\n'+str(timeCount)+'\n'
159             #loop tho molList to add data - molList = [['N','1','1','1'],['C'...],...
160             ↪ ] And element = ['N','1','1','1']
161             for element in molList:
162                 #loop through element in molList data = 'N','1','1','1'
163                 for data in element:
164                     #add to grandString and don't forget tab, return
165                     grandString=grandString+data+'\t'
166                 #end one screenshot with a return
167                 grandString=grandString+'\n'
168             #open animate.xyz for writing
169             if moleculeIsNotADuplication(grandString):

```

```

162         with open(output, 'a') as f:
163             f.write(grandString)
164             #add one to timeCount because we already write grandString
165             timeCount=timeCount+1
166             #reset all values after writing
167             atomCount=0;
168             molList=[]
169             collectionStarted=False;
170             previousGrandString=str(grandString)
171 f.close()
172 #sanity check
173 print 'Done. Extract ' + str(timeCount) + ' snapshots total.'
174

```

Plot Potential Energy of MD run

plotEnergyMD6.py script extracts potential energy and temperature of each MD frame to determine 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 have 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.

```

1  #####
2  ### Use this to plot energy vs time to see if MD has ###
3  ### run its course. Generates: csv of PE and ###
4  ### temperature vs time, pdf of the plot ###
5  #####
6
7  import matplotlib
8  matplotlib.use('Agg')
9  import matplotlib.pyplot as plt
10 import csv
11 import sys
12 import string
13
14 #call as plotEnergyMD3.py inputfile
15 #for asking what the input in terminal should be
16 try:
17     if str(sys.argv[1])=='?':
18         print '\nCall function as: plotEnergyMD.py input.log \n'

```

```

19         sys.exit()
20     else:
21         input=str(sys.argv[1])
22 except IndexError:
23     print '\n!!!Input command Error. Call function as: plotEnergyMD.py input.log
    ↪ \n'
24     sys.exit()
25 output=str(input) + '_energies.csv'
26
27 #initiate variables
28 lineBwTimeAndEnergy=0;
29 lineBwTimeAndTemp=0;
30 lineCountFromTime=0;
31 collectionStarted=False
32 foundTime=False
33 foundPE=False
34 foundTemp=False
35 firstTime=True
36 grandString='',
37
38 #####
39 #functions
40
41 #check if time is in line
42 def shouldCollectTime(line):
43     #only check if collection is in progress
44     if ' *** AT T=' in line:
45         return True;
46     else:
47         return False;
48
49 # for printing time so one can keep track of the progress
50 def printTime (thisLine):
51     lineComponents=thisLine.split();
52     timeString=str(lineComponents[3]);
53     print "Analyzing t = "+timeString+" fsec\n"
54
55 #Are we currently looking potential energy?
56 def shouldCollectPE(line,reference,currentLine):
57     #check if line bw time and energy is known - this is written as reference
58     if reference>0:
59         if currentLine==reference:
60             return True
61         else:
62             return False
63     #if reference is not known, then it needs to be found by finding string POT
    ↪ EN...

```

```

64     elif reference==0:
65         if '    POT ENERGY' in line:
66             reference=int(currentLine)
67             return True;
68         else:
69             return False;
70 #Are we currently looking Temp?
71 def shouldCollectTemp(line,reference,currentLine):
72     #check if line bw time and energy is known - reference
73     if reference>0:
74         if currentLine==reference:
75             return True
76         else:
77             return False
78     #if not then it needs to be found by searching for the string TEMPER...
79     elif reference==0:
80         if '    TEMPER' in line:
81             reference=int(currentLine)
82             return True;
83         else:
84             return False;
85
86 #once everything is found, we should write down before moving on to the next
87 ↪ 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 #open csv and prepare for writing
102 f = open(output, 'w');
103 f.write('')
104
105 #open input
106 f1=open(input)
107 print 'finding patterns...'
108 #avoid using readlines() so there'll be no problem with large files
109 for line in f1:

```

```

110  #time keyword is before PE, PE is before Temp so the search should be in this
111  ↳ 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
120          ↳ RUN'S STEP NO.= 10)
121          #this will be split to ['***', 'AT', 'T=', '10.00'...] -time =
122          ↳ element 4
123          grandString=grandString+str(lineComponents[3]);
124          foundTime=True;
125  elif not foundPE:
126      if shouldCollectPE(line,lineBwTimeAndEnergy,lineCountFromTime):
127          #append time (split) to string
128          #split line using space - sample( POT ENERGY =
129          ↳ -1.804578585E+05 KCAL/MOL)
130          #this will be split to [..., '=', '-1.804578585E+05'...] -time =
131          ↳ element 4
132          lineComponents=line.split();
133          grandString=grandString+', '+str(lineComponents[3])
134          foundPE=True
135  elif not foundTemp:
136      if not firstTime:
137          if shouldCollectTemp(line,lineBwTimeAndTemp,lineCountFromTime):
138              #append time (split) to string
139              #split line using space - sample( TEMPERATURE =
140              ↳ 349.98666547 K)
141              #this will be split to [..., '=', '-349.98666547'...] -time =
142              ↳ element 3
143              lineComponents=line.split();
144              grandString=grandString+', '+str(lineComponents[2])+'\n';
145              foundTemp=True
146          #problem with this is - the first snapshot's temperature is not given in
147          ↳ the log file
148          #set Temp to 0 to indicate the beginning
149          if firstTime:
150              grandString=grandString+',0\n';
151              #once append, turn off the boolean
152              firstTime=False
153              foundTemp=True
154          #write after all data is collected for one snapshot
155          if shouldWrite():
156              with open(output, 'a') as f:

```

```

149         f.write(grandString)
150         #reset the variables
151         lineCountFromTime=0;
152         collectionStarted=False
153         foundTime=False
154         foundPE=False
155         foundTemp=False
156         grandString=''
157         lineCountFromTime+=1;
158     #finish writing csv
159     f.close()
160
161     #plot
162     #pull out CSV
163     #use csv.reader bc csv has ',' and this automate the formatting
164     f = csv.reader(open(output))
165     #convert column to array using zip (a built in function)
166     Time, Energy, Temp = zip(*f)
167     #convert string to float
168     Time = map(float, Time)
169     Energy = map(float, Energy)
170     Temp = map(float, Temp)
171
172
173     #plot
174     x = Time
175     y1 = Energy
176     y2 = Temp
177
178     fig, ax1 = plt.subplots()
179
180     ax2 = ax1.twinx()
181     ax1.plot(x, y1, 'g-')
182     ax2.plot(x, y2, 'b-')
183
184     ax1.set_xlabel('Time (fs)')
185     ax1.set_ylabel('Potential Energy (KCal/mol)', color='g')
186     ax1.ticklabel_format(axis='y', style='sci', scilimits=(-2,2), useOffset=False)
187     ax2.set_ylabel('System Temperature (K)', color='b')
188     plt.title(r'Potential Energy vs Time for the MD File: ' + str(input))
189     #Saving to pdf gives better resolution - picture is saved to vector
190     #there is a room for improvement especially these energy plots which look very
191     ↪ mediocre
191     plt.savefig(str(input) + '_EnergyPlot.pdf', format='pdf')

```

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 CAMB3LYP 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.

```
1 #####
2 ### This is used to determined the equilibrium using ###
3 ### linear regression and an upper limit for the slope ###
4 #####
5
6 import matplotlib
7 matplotlib.use('Agg')
8 import matplotlib.pyplot as plt
9 import csv
10 import sys
11 import numpy as np
12
13 #for asking what the input in terminal should be
14 try:
15     if str(sys.argv[1])=='?':
16         print '\nCall function as: findEquilibrium.py input.log \n'
17         sys.exit()
18     else:
19         input=str(sys.argv[1])
20 except IndexError:
21     print '\n!!!Input command Error. Call function as: findEquilibrium.py
22     ↪ input.log \n'
23     sys.exit()
24 output=str(input) + '_energies.csv'
25 #This portion is the same as in plotEnergyMD6
26 #pull out CSV
27 f = csv.reader(open(output))
28 #convert column to array using zip (a built in function)
```



```

28 Time, Energy, Temp = zip(*f)
29 #convert string to float
30 Time = map(float, Time)
31 Energy = map(float, Energy)
32 Temp = map(float, Temp)
33
34 #find Equilibrium using linear regression
35 #this number control the range of time to be used in energy fluctuation
    ↪ calculation
36 minNumberOfStep=1000
37 #This is the limit above which the script will report no equilibrium is found
38 #this is from aniline32.log - 15000 to 25000
39 maxSlope=1e-4
40 #for plotting
41 slope=[]
42 print 'Finding equilibrium using minimum number of steps = '+str(minNumberOfStep)
    ↪ '+' and top limit of acceptable slope = '+str(maxSlope)
43 try:
    #for looping
44     size=len(Time)
45     x=Time[0:minNumberOfStep]
46     y=Energy[0:minNumberOfStep]
47     #for using in loop
48     #set thio a high value - it can be any number bc we will replace it with the
    ↪ lowest slope value found in loop
49     lowestSlopeValue=1e5
50     #Same - this will be replaced
51     indexOfLowestSlope=-1
52     for i in range(0,size-1-minNumberOfStep):
53         print 'Finding equilibrium from t= '+str(Time[i])+' to
    ↪ '+str(Time[i+minNumberOfStep-1])
54         #poly fit is basically a linear fit - m=slope, b=y_intersect
55         m,b = np.polyfit(x, y, 1)
56         #for plotting - append to array of existing slope values
57         slope.append(m)
58         print 'slope = ' +str(m)
59         #take absolute value and see which interval does not fluctuate the least
60         if abs(m)<=abs(lowestSlopeValue):
61             lowestSlopeValue=float(m)
62             indexOfLowestSlope=int(i)
63         #this is similar to queue structure - room for improvement is to make x
    ↪ and y arrays into actual queues
64         #else remove the head and add next tail
65         del x[0]
66         x.append(Time[minNumberOfStep+i])
67         del y[0]
68         y.append(Energy[minNumberOfStep+i])
69

```

```

70 #if x or y does not have enough element (minNumberOfStep) then report error
71 #room for improvement - move this up top instead of having a long try
72 except IndexError:
73     print 'There is not enough data to determine the equilibrium'
74 #if lowestSlopeValue pass the top limit then report
75 if abs(lowestSlopeValue)<=abs(maxSlope):
76     print 'Found best equilibrium starting from '+str(Time[indexOfLowestSlope])+
        ↳ to '+str(Time[indexOfLowestSlope+minNumberOfStep-1])+ with slope =
        ↳ '+str(lowestSlopeValue)
77 #if not then say so
78 else:
79     print 'Equilibrium is not yet reach.'
80     print 'The current limit is at '+str(maxSlope) +' kcal/mol/fs and the
        ↳ lowest value of slope = '+str(lowestSlopeValue)
81
82 #plot slope vs time
83 x=slope
84 #align time with slope
85 y=Time[0:size-1-minNumberOfStep]
86 plt.plot(y,x)
87 plt.xlabel('Starting Time (fs)')
88 plt.ylabel('Slope (KCal/mol/fs)')
89 plt.ticklabel_format(axis='y', style='sci', scilimits=(-2,2), useOffset=False)
90 plt.title('Slope vs Time for the MD File: ' + str(input))
91 plt.savefig(str(input) + '_SlopePlot.pdf', format='pdf')

```

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 Nathan 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.

```

1 #####
2 ### This script create a folder of inp for TDDFT ###
3 #####
4
5 import os
6 import sys
7
8 #call as fincut.py input.log startTime stopTime timePerFrame
9 #this will run from starting startTime+timePerFrame to stopTime
10 #for example 15010-25000 if input is 15000, 25000

```

```

11
12 #this dict is for generating atomic number from Acronym
13 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}

14
15 #if not sure use ? to ask
16 try:
17     if str(sys.argv[1])=='?':
18         print '\nCall function as: fincut.py input.log
            ↪ numberOfSoluteAtoms numberOfSolventAtoms
            ↪ numberOfSolventMolecules startTime stopTime timePerFrame\n'
19         sys.exit()
20 except IndexError:
21     print '\n!!!Input command Error. Call function as: fincut.py input.log
            ↪ numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules
            ↪ startTime stopTime timePerFrame\n'
22     sys.exit()
23
24 #Call as fincut.py input.log numberOfSoluteAtoms numberOfSolventAtoms
    ↪ numberOfSolventMolecules startTime stopTime timePerFrame
25 try:
26     input=str(sys.argv[1])
27     numberOfSoluteAtoms=int(sys.argv[2])
28     numberOfSolventAtoms=int(sys.argv[3])
29     numberOfSolventMolecules=int(sys.argv[4])
30     startTime =int(sys.argv[5])
31     stopTime =int(sys.argv[6])
32     timePerFrame=int(sys.argv[7])
33 except IndexError:

```

```

34     print '\n!!!Input command Error. Call function as: fincut.py input.log
        ↳ numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules
        ↳ startTime stopTime timePerFrame\n'
35     sys.exit()
36
37     #should collect data when 1) time within interval specified
38     #2) they are x y z line - not line 1 and 2
39     def shouldCollect(numberOfLine):
40         frame=int(numberOfLine/totalNumberOfLineInOneSet)
41         time=frame*timePerFrame
42         #if within range
43         if time>startTime and time<=stopTime:
44             #0 and 1 (line 1 and 2) in each frame in xyz are not useful
45             if numberOfLine%totalNumberOfLineInOneSet!=0 and
                ↳ numberOfLine%totalNumberOfLineInOneSet!=1:
46                 return True
47             else:
48                 return False
49         else:
50             return False
51
52     #condition for writing to each inp
53     def shouldWrite(numberOfLine):
54         #should write when last line of a frame is read
55         if numberOfLine%totalNumberOfLineInOneSet==totalNumberOfLineInOneSet-1 :
56             return True
57         else:
58             return False
59
60     #write FRAGNAME=H2ODFT ! 1
61     def shouldWriteFragmentHeader(numberOfLineInSolvent):
62         #should write before writing geometry of solvent molecules
63         if numberOfLineInSolvent%numberOfSolventAtoms==0 :
64             return True
65         else:
66             return False
67
68     #create format of GAMESS inp
69     def insertAtomicNumberInto(line):
70         lineSplit=line.split()
71         lineSplit.insert(1,str(atomicNumber[lineSplit[0]]))
72         return
            ↳ lineSplit[0]+' \t'+lineSplit[1]+' \t'+lineSplit[2]+' \t'+lineSplit[3]+' \t'+li
73     #create format of GAMESS inp O -> O1, H-> H2...
74     def insertCountsInto(line, count):
75         lineSplit=line.split()
76         lineSplit[0]=lineSplit[0]+str(count)

```

```

77         return
78         ↪ lineSplit[0]+'\\t'+lineSplit[1]+'\\t'+lineSplit[2]+'\\t'+lineSplit[3]+'\\n'
79
80 #even tho input is received written in .log - it will ultimately use .xyz created
81 ↪ by 3dExtract for efficiency
82 #this can be confusing - room for improvement
83 if input.endswith('.log'):
84     input = input[:-4]
85
86 #Path for storing input files
87 path = os.getcwd()
88 inputPath=path+'\\'+input+'InputFiles'
89 #create folders if not already done
90 if not os.path.exists(inputPath): os.makedirs(inputPath)
91
92 #initiate variables
93 grandString=[]
94 lineNumber=0
95 numberOfAllSolventsAtoms=numberOfSolventAtoms*numberOfSolventMolecules
96 totalNumberOfAtoms=numberOfSoluteAtoms+numberOfAllSolventsAtoms
97 totalNumberOfLineInOneSet=totalNumberOfAtoms+2
98
99 #if no xyz in folder - ask for it
100 #if no 3dExtract.py - print...
101 #if there is, call it from here + print s'th
102 #else create one?
103 #if there is, then proceed
104 #this is very confusing - if there is xyz but it's not updated when it's of no
105 ↪ use - room for improvement
106
107 #open input.xyz
108 try:
109     f=open(input+'.xyz')
110 except IOError:
111     print '\\nThere is currently no xyz file named '+input
112     print 'trying to call 3dExtract'
113     try:
114         os.system('python 3dExtract4.py'+ ' '+input+'.log'+
115             ↪ '+str(numberOfSoluteAtoms)+' '+str(numberOfSolventAtoms)+'
116             ↪ '+str(numberOfSolventMolecules))
117     except IOError:
118         print '\\n Error. There is no 3dExtract to call. Try copying 3dExtract
119             ↪ here. \\n'
120         print 'Process terminated abnormally'
121         sys.exit()
122     f=open(input+'.xyz')

```

```

118 #since we are reading from xyz - using readlines() is okay
119 numberOfAllAtoms=int(f.readline().strip())
120 #enumerate gets data in line - line and line index - n
121 for line in f:
122     lineNumber+=1
123     #check if the line should be written
124     if shouldCollect(lineNumber):
125         grandString.append(line)
126         #if about to write -create file with this name
127         if shouldWrite(lineNumber):
128             frame=int(lineNumber/totalNumberOfLineInOneSet)
129             time=frame*timePerFrame
130             f1 = open(inputPath+'/' +input+'_'+str(time)+'.inp','w')
131             #write header
132             headerString=""" $CONTRL SCFTYP=RHF TDDFT=EXCITE DFTTYP=CAMB3LYP
↪ RUNTYP=ENERGY
133             ICHARG=0 MULT=1 COORD=UNIQUE MAXIT=200 $END
134             !TDDFT requires lots of memory space
135             $SYSTEM MWORDS=200 MEMDDI=250 $END
136             $SCF DIRSCF=.T. $END
137             $TDDFT NSTATE=5 TPA=.f. $END
138             $BASIS GBASIS=N311 NGAUSS=6 NDFUNC=2 NPFUNC=1
139             DIFFSP=.TRUE. DIFFS=.TRUE. POLAR=POPN311 $END
140             $DATA\n"""+input+' at t= '+str(time)+'\nC1 1\n'
141             print 'Making input file for 'input+' at t= '+str(time)
142             #write the header
143             f1.write(headerString)
144             #write one line by one - before writing we need to add atomic number
↪ in using the function defined above
145             for eachLine in grandString[:numberOfSoluteAtoms]:
146                 eachLine=insertAtomicNumberInto(eachLine)
147                 f1.write (eachLine)
148             #end solute and go to solvent
149             stringBwSoluteAndSolvent=""" $END\n\n $EFRAG\nCOORD=CART
↪ POSITION=OPTIMIZE \n"""
150             f1.write(stringBwSoluteAndSolvent)
151             #write fragments
152             #for iteration
153             i=0
154             for numberOfLineWithInSolvents, eachLine in
↪ enumerate(grandString[numberOfSoluteAtoms:]):
155                 #is header needed - if so write
156                 if shouldWriteFragmentHeader(numberOfLineWithInSolvents):
157
↪ fragmentsNumber=numberOfLineWithInSolvents/numberOfSolventAtoms+1
158                 f1.write("FRAGNAME=H2ODFT ! "+str(fragmentsNumber)+'\n')
159                 #write each line in fragments

```

```

160         atomNumberInFragments=i%numberOfSolventAtoms+1
161         eachLine=insertCountsInto(eachLine, atomNumberInFragments)
162         f1.write (eachLine)
163         i+=1
164         #close and reset
165         f1.write (" $END\n")
166         grandString=[]
167         f1.close()
168
169     #create megaio for gmssub input
170     allFiles = os.listdir(inputPath)
171     program = 'gmssub'
172     processors = '32'
173
174     #create the write file - like gmssub aniline32.inp aniline32.log -l 10g=true
175     f = open('megaio_'+input+'.txt','w')
176     #write each item
177     for item in allFiles:
178         #for GAMESS output naming
179         outputName=str(item)[:4]+'log'
180         f.write(program + ' %s %item + processors + ' '+outputName+' -l 10g=true\n')
181     f.close()

```

Pull Excited State Energies from TD-DFT Log Files

postMDDDataPull2.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...

```

1  #####
2  ### Use this pull out excited state energies and dipole ###
3  ### moments from all the files in specified folder      ###
4  ### 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: postMDDDataPull2.py inputDirectory
16             ↳ NumberOfExcitedStateEnergies\n'
17         sys.exit()
18 except IndexError:

```

```

18     print '\n!!!Input command Error. Call function as: postMDDDataPull2.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: postMDDDataPull2.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
60         fileIsComplete=False
61         #find if the file is complete

```



```

62     # use enumerate and reverse
63     for i, line in reversed(list(enumerate(lines))):
64         if ' STATE # 1 ENERGY = ' in line:
65             startingLine=i
66             fileIsComplete=True
67             break
68     # now start collecting data
69     if fileIsComplete:
70         print 'Harvesting data from file named: '+fileName
71         counter=0
72         #loop
73         for n, line in enumerate(lines):
74             #time
75             if line.startswith(' RUN TITLE'):
76                 nextLine=lines[n+2]
77                 time=str(nextLine.split()[3])
78                 f1.write(time)
79                 f2.write(time)
80             #energy
81             if line.startswith(' STATE # '):
82                 E=line.split()[5]
83                 nextLine=lines[n+1]
84                 f=nextLine.split()[3]
85                 f1.write(','+E+','+f)
86                 counter+=1
87             #stop collecting
88             if (counter==numberOfExcitedStates):
89                 f1.write('\n')
90                 counter=0
91             #dipole
92             if line.startswith(' SUMMARY OF
93                 ↳ TDDFT RESULTS'):
94                 #5 lines are from 'SUMMARY...' to ' 1 A ...'
95                 ↳ which starts to contain dipole moments
96                 for linesAhead in
97                     ↳ range(5,5+numberOfExcitedStates):
98                     lineContainingDipoles=lines[n+linesAhead]
99                     lineSplit=lineContainingDipoles.split()
100                     #locations of dipole in line
101                     [X,Y,Z]=[lineSplit[4], lineSplit[5],
102                             ↳ lineSplit[6]]
103                     f2.write(','+X+','+Y+','+Z)
104                 f2.write('\n')

```

GAMESS inputs

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 ± 25 degree Celsius. Solvent boundary potential is also activated using default Sforce value ($1.0 \text{ kcal} \cdot \text{mol}^{-1} \cdot \text{\AA}^{-2}$), 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)

```
1  # run type = MD, with functional = CAMB3LYP, COORD = UNIQUE is important
2  $CONTRL SCFTYP=RHF RUNTYP=MD COORD=UNIQUE
3      DFTTYP=CAMB3LYP MAXIT=200 ICHARG=0 MULT=1 $END
4  # MD is recording every 10 frames with default 1 frame =10 fs
5  # 25 degree celcius, RSTEMP is on for keeping the temp ~ +/-25
6  # ssbp is on with default SForce value and radius estimated from prepareMD2.py
7  $MD KEVERY=10 PROD=.T. NVTNH=2 MBT=.T. MBR=.T.
8      BATHT=298 RSTEMP=.T. DTEMP=25 NSTEPS=50000
9      SSBP=.T. SFORCE=1.0 DROFF=12.0632116659 $END
10 #####1#####
11 #####
12 # dispersion correction is off
13 $DFT DC=.F. $END
14 # memory requested at each node =1000 million words
15 # memory reserved for communication = 1000 million words
16 $SYSTEM MWORDS=1000 MEMDDI=1000 $END
17 $SCF DIRSCF=.T. $END
18 # Basis set = 6-31+(2d,p)
19 $BASIS GBASIS=N31 NGAUSS=6 NDFUNC=2 NPFUNC=1
20     DIFFS=.TRUE. POLAR=POPN31 $END
21 # solute geometry - C1 1 = symmetry data
22 $DATA
23 MD INPUT for aniline32
24 C1 1
25 #####2#####
26 N      7.0      -2.3128100000      -0.0046000000      -0.0894530000
27 C      6.0      -0.9197160000      -0.0031280000      -0.0360090000
28 C      6.0      -0.2076150000      1.2004070000      -0.0355880000
29 .
30 .
31 H      1.0      -2.7544730000      0.8437050000      0.2369240000
32 H      1.0      -2.7570450000      -0.8248640000      0.2993670000
33 #####
```

```

34 $END
35
36 # solvent geometry in EFP1 (EFP2 is still not available)
37 #####3#####
38 $EFRAG
39 COORD=CART POSITION=OPTIMIZE
40 FRAGNAME=H2ODFT ! 1
41 O1      1.7760990000      4.8390610000      -2.1049530000
42 H2      0.9224740000      4.4173920000      -2.2628490000
43 H3      2.4128200000      4.1148590000      -2.1441810000
44 FRAGNAME=H2ODFT ! 2
45 O1      3.6783070000      3.8351060000      0.8717800000
46 H2      3.6020030000      4.1116030000      1.7932670000
47 H3      3.4138960000      4.6126330000      0.3648680000
48 .
49 .
50 FRAGNAME=H2ODFT ! 32
51 O1      3.7691430000      -1.4091250000      -4.0319510000
52 H2      2.8756120000      -1.5562470000      -4.3656710000
53 H3      3.6686840000      -1.3995500000      -3.0721400000
54 $END
55 #####

```

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.

```

1 .
2 .
3 #time = 38960 fs
4 ===== MD DATA PACKET =====
5 NAT=      14 NFRG=      32 NQMMM=      0
6 TTOTAL=    38960.00 FS   TOT. E=    -180710.543716 KCAL/MOL
7 POT. E=    -180783.917894 KCAL/MOL BATHT=      298.000000
8 KIN. E=      73.374179 TRANS KE=  44.012966 ROT KE=  29.361213 KCAL/MOL
9 ----- QM PARTICLE COORDINATES FOR $DATA GROUP -----
10 #####2#####
11 N          7.0      3.8554852781      -1.5814882196      -3.7440432660
12 C          6.0      2.9445799802      -1.6872198850      -4.7513152920
13 .
14 .

```

```

15  H          1.0          3.8827088558          -0.7337182877          -3.1704671239
16  #####
17  ----- EFP PARTICLE COORDINATES FOR $EFRAG GROUP -----
18  #####3#####
19  $EFRAG
20  COORD=CART POSITION=OPTIMIZE
21  FRAGNAME=H2ODFT ! 1
22  O1          0.4899097683          5.6815060052          1.3332597175
23  H2          -0.1774491436          5.6953737679          2.0005813955
24  H3
25  .
26  .
27  FRAGNAME=H2ODFT ! 32
28  O1          1.8397452656          -1.0633683830          1.4432099580
29  H2          2.6999380304          -1.1499324935          1.8219625372
30  H3          1.3989608369          -0.3812464183          1.9241435987
31  $END
32  #####
33  GRADIENT DATA (NOT USED BY RESTARTS)...
34  FRAGMENT # 1 H2ODFT
35  .
36  .
37  ----- RESTART VELOCITIES FOR $MD GROUP -----
38
39  #####1#####
40  $MD READ=.TRUE. MBT=.FALSE. MBR=.FALSE. TTOTAL= 3.90E-11
41  MDINT= VVERLET DT= 0.10E-14 NVTNH= 2 NSTEPS= 11040
42  RSTEMP=.T. DTEMP= 25.00 LEVER= 50000
43  RSRAND=.F. NRAND= 1000 NVTOFF= 0 JEVERY= 10
44  PROD=.T. KEVERY= 10 DELR= 0.020
45  Batht(1)=298.00
46  SSBP=.T. SFORCE= 1.0 DROFF= 12.1*****
47  TVELQM(1)= ! QM ATOM TRANS. VELOCITIES (BOHR/PS) !
48  -1.436664655E+00 -1.014912632E+00 1.731488079E+01
49  .
50  .
51  -4.714191860E+00 -3.065697154E+00 1.306355299E+01
52  TVEL(1)= ! EFP TRANSLATIONAL VELOCITIES (BOHR/PS) !
53  -9.879998843E+00 -1.286351783E+01 -1.625361337E-01
54  .
55  .
56  4.573922683E+00 9.602851076E+00 6.875495095E+00
57  QUAT(1)= ! EFP QUATERNIONS !
58  6.658493597E-01 -7.408965451E-01 1.545516153E-02 8.647587874E-02
59  .
60  .
61  8.096376474E-01 4.753557554E-01 1.423717372E-01 3.134550593E-01

```

```

62 RVEL(1)=      ! EFP ANGULAR VELOCITY (RAD/PS) !
63   1.392338986E+01 -1.145718732E+01 -1.039863492E+01
64 .
65 .
66   -1.217796312E+01  2.229922184E+00  2.270460047E+01
67 QUAT1D(1)=      ! EFP QUATERNION 1ST DERIV. !
68   4.237107071E+12  4.149735474E+12  8.424472510E+12  1.422917746E+12
69 .
70 .
71   5.909828315E+12 -1.125106858E+13 -8.408659179E+11  2.179439222E+12
72 QUAT2D(1)=      ! EFP QUATERNION 2ND DERIV. !
73  -6.427981808E+26 -4.203604186E+26 -7.365308723E+25  1.102236012E+26
74 .
75 .
76   8.379527677E+26  4.204660564E+26 -1.448608909E+27 -2.676740098E+27
77 $END
78 #####
79
80 #time = 38970 fs
81 ===== MD DATA PACKET =====
82 NAT=      14 NFRG=      32 NQMMM=      0
83 TTOTAL=    38970.00 FS    TOT. E=    -180715.284973 KCAL/MOL
84 POT. E=    -180783.048242 KCAL/MOL  BATHT=      298.000000
85 KIN. E=      67.763269  TRANS KE=  41.754223  ROT KE=  26.009046 KCAL/MOL
86 ----- QM PARTICLE COORDINATES FOR $DATA GROUP -----
87 N          7.0          3.8518322659          -1.6019379232          -3.6477010146
88 .
89 .

```

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)

```

1  # run type = [excitation] energy, with functional = CAMB3LYP, and TDDFT
2  $CONTRL SCFTYP=RHF TDDFT=EXCITE DFTTYP=CAMB3LYP RUNTYP=ENERGY
3      ICHARG=0 MULT=1 COORD=UNIQUE MAXIT=200 $END
4  #TDDFT requires lots of memory space
5  # memory requested at each node =1000 million words
6  # memory reserved for communication = 1000 million words
7  ↪
7  $SYSTEM MWORDS=200 MEMDDI=250 $END
8  #activate direct SCF calculation
9  $SCF DIRSCF=.T. $END
10 # find 5 excited states - the current setting is purely driven by its lower cost

```

```

11  # Previous experience shows that 10 states gives only a few strong peak.
    ↪
12  $TDDFT NSTATE=5 TPA=.f. $END
13  # Basis set = 6-311++(2d,p)
14  $BASIS GBASIS=N311 NGAUSS=6 NDFUNC=2 NPFUNC=1
15      DIFFSP=.TRUE. DIFFS=.TRUE. POLAR=POP311 $END
16  # solute geometry - C1 1 = symmetry data
17  $DATA
18  aniline32 at t= 15010
19  C1 1
20  N      7.0      2.4008547653      5.9114221893      -1.1412310058
21  C      6.0      1.9371475177      5.9223533811      -2.4157851823
22  C      6.0      0.6209366009      6.2361805033      -2.7812041720
23  .
24  .
25  H      1.0      1.7323956480      5.6459040114      -0.4329519914
26  H      1.0      3.3564486514      5.6102990678      -1.0242941608
27  $END
28
29  # solvent geometry in EFP1 (EFP2 is still not available)
30  $EFRAG
31  COORD=CART POSITION=OPTIMIZE
32  FRAGNAME=H2ODFT ! 1
33  O1      2.335993939511      3.751856628604      1.427418842826
34  H2      1.439322965739      3.833168541986      1.710699607266
35  H3      2.854351938959      3.613582975500      2.203990690907
36  FRAGNAME=H2ODFT ! 2
37  O1      3.266753260182      2.613267395174      3.808546039622
38  H2      3.514552920456      1.768271678250      3.468757839439
39  H3      3.822697636667      2.780022240614      4.552855871622
40  .
41  .
42  FRAGNAME=H2ODFT ! 32
43  O1      -0.041331901955      -3.906598195206      1.282021099515
44  H2      -0.790424236756      -4.406259423929      1.565001283174
45  H3      -0.374908444181      -3.111491773717      0.898079798177
46  $END

```

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) 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.
- (4) 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.
- (5) 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.
- (6) 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.
- (7) 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.
- (8) Kliegman, S.; Eustis, S. N.; Arnold, W. a.; McNeill, K. Experimental and theoretical in-

- sights into the involvement of radicals in triclosan phototransformation. *Environmental science & technology* **2013**, *47*, 6756–63.
- (9) 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.
 - (10) Dykstra, C. E.; Frenking, G.; Kim, K. S. *Elsevier*; 2011.
 - (11) Hirayama, S.; Steer, R. P. Understanding the Theory and Practice of Molecular Spectroscopy: The Effects of Spectral Bandwidth. *Journal of Chemical Education* **2010**, *87*, 1344–1347.
 - (12) Klán, P.; Wirz, J. *Photochemistry of organic compounds: From concepts to practice*; John Wiley & Sons, 2009.
 - (13) 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.
 - (14) Cossi, M.; Barone, V. Solvent effect on vertical electronic transitions by the polarizable continuum model. *The Journal of Chemical Physics* **2000**, *112*, 2427.
 - (15) Barone, V.; Polimeno, A. Integrated computational strategies for UV/vis spectra of large molecules in solution. *Chem. Soc. Rev.* **2007**, *36*, 1724–1731.
 - (16) 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.
 - (17) Tomasi, J.; Mennucci, B.; Cammi, R. Quantum mechanical continuum solvation models. *Chemical Reviews* **2005**, *105*, 2999–3093.

- (18) 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.
- (19) 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.
- (20) 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.
- (21) 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.
- (22) Wiberg, K. B. Basis set effects on calculated geometries: 6-311++G** vs. aug-cc-pVDZ. *Journal of Computational Chemistry* **2004**, *25*.
- (23) 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.
- (24) Beglov, D.; Roux, B. Finite representation of an infinite bulk system: Solvent boundary potential for computer simulations. *The Journal of Chemical Physics* **1994**, *100*.
- (25) Plugatyr, A.; Svishchev, I. M. The hydration of aniline: Analysis of spatial distribution functions. *The Journal of chemical physics* **2009**, *130*, 114509.