

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

Trerayapiwat K.* and Eustis S.*

Department of Chemistry, Bowdoin College, Brunswick, ME

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

Abstract

Understanding excitation from ground state to the singlet excited state through simulating absorption spectra of a molecule is essential to predicting the rate of the photoreaction. Excitation energies and oscillator strengths were calculated using different theories and methods. Among all theories, a new approach was selected to model the photon absorption: Molecular Dynamics Time Dependent Density Functional Theory (MD-TDDFT). 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 TDDFT with w-B97X-D, CAMB3LYP, or M06-2X functional. As a theoretical benchmark, OEMCCSD calculation was carried out with optimized geometry from using using 6-311++G** basis set and implicit water model implemented using Polarizable Continuum Model (PCM). The computed physical properties from MD-TDDFT and OEMCCSD were then compared with data from experimental absorption spectra to

evaluate the accuracy of the two 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. The more accurate method would be applied to triclosan and other water contaminants to predict the rate of their photodegradations in the environment.

Introduction

Environmental Photochemistry and Excited State Energies calculation

Micropollutants are pollutants whose concentrations are individually low, but combined have an effect that is hard to predict. The studies of particular micropollutant specie are hard to conduct because other species interfere in analytical reactions. Triclosan, a micropollutant, has been used as an anti-bacterial agent in household soap and health care products. Under sunlight, Triclosan decomposes to Dioxins and PCBs, well-known carcinogens.¹ Previously, computational studies of Triclosan in the excited states were carried out by Soren N. Eustis² and Nathan Ricke.³ While excited states are important to understanding photochemical reaction, excitation from ground states by photons to excited states is equally important to understand complete reaction mechanism. There are currently no prior studies on how to quantitatively calculate excited state energies of organic molecule in water allowing a systematic approach to develop computational model to be studied. 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.

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 sol-

vation models remains a big challenge.⁴ 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 Potential Continuum Model (PCM).⁵ Instead of explicitly handling each solvent molecules 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.⁶ Furthermore, implicit solvent model also neglects hydrogen-bonding as it assumes implicit implementation in dispersion forces and electrostatics.⁷ Especially in calculating excited state energies, an accurate solvent model should be used.⁸ 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.^{9,10} This model is chosen to implement explicit solvent in calculating excited states energies.

In modeling organic solute in aquatic environment, the solute, the appropriate number of water molecules to be included as EFP in the model has never been evaluated. Too many water means expensive computational cost. Too few water will not fully model solvating shells around the solute. Binary system will be used to model how many water molecule is needed to fully solvate the solute molecules: 2, 4, 8, 16... Once excited energies for each system is calculated, the results will be compared with experimental value to evaluate how many water is needed before determining on which functional out of three choices should be chosen to achieve the most accurate computational model.

Computational Models: Theories, Basis Sets, and Functionals

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

computational cost.¹¹ Implementing EFP solvent model, TDDFT can be used to accurately calculate excited state energy of acetone in water.¹⁰ 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.¹² 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 averaged the calculated energies of each excited state to arrive at a final excited states energy.¹²

According to previous basis set studies, while having roughly the same computational cost, an average-sized basis set 6-311++(2d,p) performs better than aug-cc-pVDZ (ACCD).^{13,14} 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++(2d,p) only deviate 220-470 cm^{-1} . Hoping to most accurately calculate the excited energies, 6-311++(2d,p) basis set is chosen to run TDDFT after MD run. In running MD, a smaller basis set 6-31+(2d,p) will be used in order to cut computational cost. The decision comes 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.¹⁴ PBE0 will also be used.

Method

Molecules - aniline - then para-methoxy m-methoxyacetophenone... Triclosan

discuss # of water

Preliminary Results and Discussions

Determining the number of water

TDDFT 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. 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 1d. The consistently low fluctuation indicates the start of equilibrium at 15000 fs. 1000 frames or 10000 fs of MD geometries were used to calculate the excitation energies in TDDFT run. 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.¹⁵

Table 1: Wavelength and Oscillator Strength from MD-TDDFT 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

The excited state energies and its oscillator strength are tabulated in table 1. When compared with aniline’s UVVIS spectra, as in figure 2, there are several problems. Firstly, the calculated value at 246 nm does not appropriately 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.

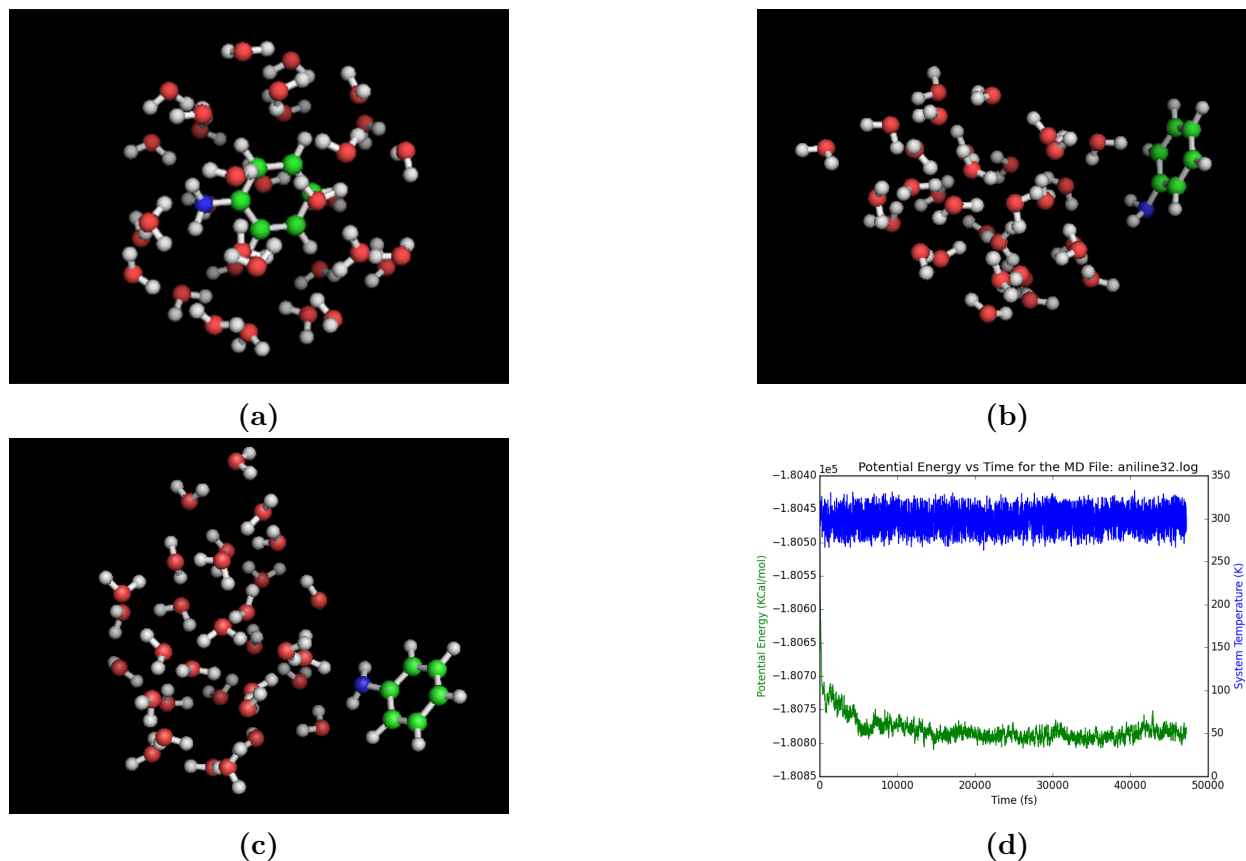


Figure 1: Molecular Dynamics run of aniline in 32 explicit solvating water molecules. Notice that at equilibrium, aniline molecule comes outside of the water sphere. Albeit hydrogen bond being clearly established, lack of total submersion in water means 32-water does not fully solvate the aniline molecule 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.

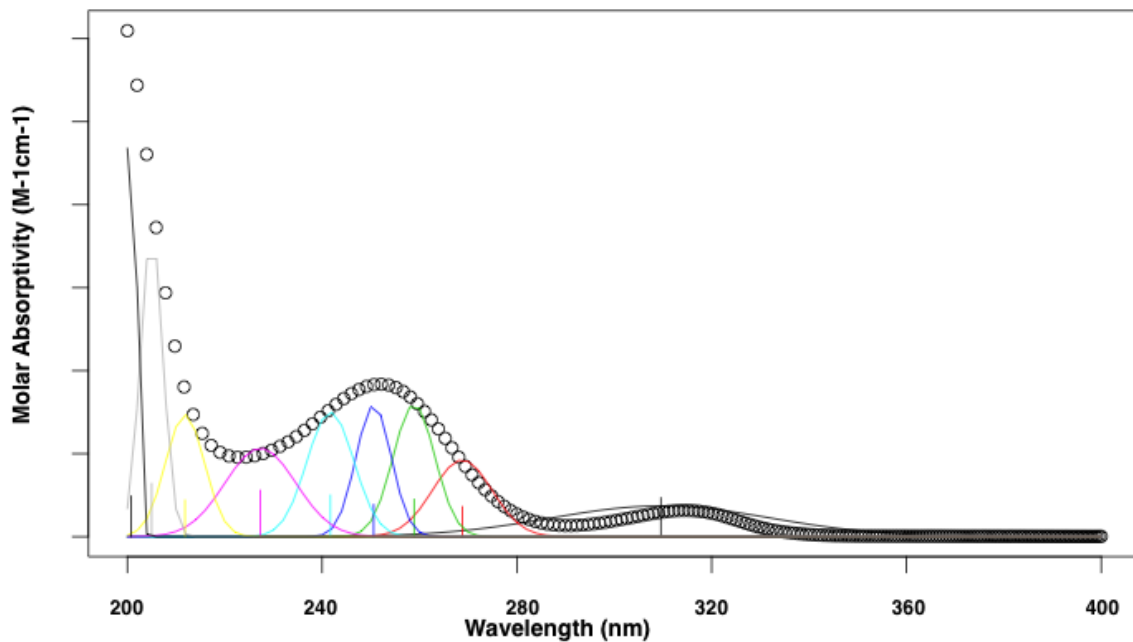


Figure 2: Experimental UVVIS spectra. Gaussian plots are fitted under the curve to find oscillator strength underlying the curve. Note here that data starts from 200 nm to 400 nm. Wavelength oscillator strength of underlying gaussians are reported in table.

Table 2: Wavelength and Oscillator Strength calculated from experimental UV-VIS spectrum using Bayesian probability (see appendix for R code).

Wavelength (nm)	Oscillator Strength
241.53	1808.0
234.52	2251.3
228.95	1939.4
223.28	2506.5
214.67	2795.1
206.10	2208.4
202.53	3202.8
200.39	2447.5

Appendix

Python Scripts

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 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 `postMDDataPull2.py` is designed to pull thousands of geometries and generate GAMESS input files for TDDFT 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.

Preparing MD Input Files

This script does two things. First (line 35-84), it calculates appropriate radius for solvent boundary potential. Some time and effort were spent on figuring out what the radius should be without empirically guess it. A simple model is proposed: At most solute will rotate around its outmost solute atom. This radius, in the code, is called solute radius. The other radius is solvent radius, its the distance between the outmost solvent atom to the solute's CG. 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 is required for GAMESS input files, so 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
23 try:
24     input=str(sys.argv[1])
25     numberOfSoluteAtoms=int(sys.argv[2])
26     numberOfSolventAtoms=int(sys.argv[3])
27     numberOfSolventMolecules=int(sys.argv[4])
28 except IndexError:
29     print '\n!!!Input command Error. Call function as: prepareMD.py input.xyz
30         ↳ numberOfSoluteAtoms numberOfSolventAtoms numberOfSolventMolecules \n'
31     sys.exit()
32 #generate output name
33 if input.endswith('.xyz'):
34     output = input[:-4]+' .inp'
35 #for safety - at worst the output will not overwrite the input
36 else:
37     output=input+' .inp'
38
39
40 #part one
41 #This part is for finding ssbp radius for inout file
42 #enumerate gets data in line - line and line index - n
43 radiusInSolute=0.0
44 radiusInSolvent=0.0
45 avgX=0.0
46 avgY=0.0
47 avgZ=0.0
48 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
53     ↪ line
54     if lineNumber>2:
55         #x y z
56         lineSplit=line.split()
57         X.append(float(lineSplit[1]))
58         Y.append(float(lineSplit[2]))
59         Z.append(float(lineSplit[3]))
60 #for looping through array below
61 size=len(X)
62 #find a CG for solute atoms
63 avgX=sum(X[:numberOfSoluteAtoms-1])/numberOfSoluteAtoms
64 avgY=sum(Y[:numberOfSoluteAtoms-1])/numberOfSoluteAtoms
65 avgZ=sum(Z[:numberOfSoluteAtoms-1])/numberOfSoluteAtoms
66 #looping to find radius of each atoms in relative to solute's CG
67 #also find the maximum value of them
68 for i in range(0,size):
69     d=((X[i]-avgX)**2+(Y[i]-avgY)**2+(Z[i]-avgZ)**2)**0.5
70     if i<numberOfSoluteAtoms:
71         if radiusInSolute<d:
72             radiusInSolute=d
73     else:
74         if radiusInSolvent<d:
75             radiusInSolvent=d
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

```

```

88  numberOfAllSolventsAtoms=numberOfSoventAtoms*numberOfSolventMolecules
89  fragmentNumber=1;
90  atomLabel=1
91
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)
            ↳ to num+2)
118         #it's num+2 bc the range will go to num+1
119         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. This script allows geometries to be extracted 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
```

```

34 previousGrandString='',
35 collectionStarted=False
36 time='',
37 #1 is for cartesian line (useless), then 1 in 3(n+1) is for fragment H2O line
38   ↪ (also useless)
39 numberOfLinesToBeCollected=numberOfSoluteAtoms+1+numberOfSolventMolecules*(numberOfSolventAtoms
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;

```

```

78  #Even now I still don't understand why GAMESS duplicate system geometry for a
    ↳ step twice in the log file
79  #This is written to prevent duplication of geometry in the xyz-movie file
80  def moleculeIsNotADuplication(currentMoleculeToBeWritten):
81      # to compare previously stored geometry and a new one is tricky bc each
        ↳ 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
122         ↳ molList!
123         #if you don't realize six asterisk then you should go back up - we do
124         ↳ this so we can remove element along the way without messing up the
125         ↳ index
126     for line in list(molList):
127         #if line has 4 elements then it's a coordinate from solvent fragment
128         ↳ - we have to drop number behind atom - 01 to 0
129         if len (line) == 4:
130             #store string
131             oldString = line[0]
132             #replacement string
133             newString=''
134             #loop to check if it's a alphabet or not
135             for character in range(len(oldString)):
136                 #do substring of 1 character
137                 subString = oldString[character:character+1]
138                 #check if it's an alphabet - yes? then add to newString
139                 if subString.isalpha():
140                     newString = newString + subString
141                 #replace '01' with '0'
142                 line[0]=newString
143             #if it's 5 then it's solute coordinate - we have to get rid of atomic
144             ↳ number behind atomic representation
145             elif len(line) == 5:
146                 # 'N 7.0 ...' will become 'N ...'
147                 del line[1]
148             #the rest are crap - just remove it out of the line
149             else:
150                 #there's a reason why this is remove - not del - since we are
151                 ↳ iterating if we delete using index we are gonna be screwed
152                 molList.remove(line)
153             #this is for if we have an incomplete file or inconsistant number of
154             ↳ atoms we should only use the one before and break for loop without
155             ↳ appending to grandString
156         if len(molList) != atomCount:
157             print 'error'
158         #xyz file has a format that we need atomCount at the top followed by
159         ↳ snapshot number(timeCount) on the next line before adding any
160         ↳ coordinates
161         grandString=grandString+str(atomCount)+'\n'+str(timeCount)+'\n'
162         #loop tho molList to add data - molList = [['N','1','1','1'],['C'...],...
163         ↳ ] And element = ['N','1','1','1']
164         for element in molList:
165             #loop through element in molList data = 'N','1','1','1'

```



```

155         for data in element:
156             #add to grandString and don't forget tab, return
157             grandString=grandString+data+'\t'
158             #end one screenshot with a return
159             grandString=grandString+'\n'
160         #open animate.xyz for writing
161         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

```

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 degree Celsius with \pm 25 degree Celsius. #####n##### are for restarting MD in case the calculation abruptly ends (see next section).

```

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  C      6.0      1.1800230000      1.1968760000      -0.0402490000
30  C      6.0      1.8885400000      0.0002120000      -0.0451380000
31  C      6.0      1.1826030000      -1.1982380000      -0.0439250000
32  C      6.0      -0.2049580000      -1.2053990000      -0.0404060000
33  H      1.0      -0.7444290000      -2.1464580000      -0.0434230000
34  H      1.0      1.7166170000      -2.1417620000      -0.0464910000
35  H      1.0      2.9715860000      0.0014800000      -0.0479970000
36  H      1.0      1.7114190000      2.1418420000      -0.0377940000
37  H      1.0      -0.7497420000      2.1399280000      -0.0298190000
38  H      1.0      -2.7544730000      0.8437050000      0.2369240000
39  H      1.0      -2.7570450000      -0.8248640000      0.2993670000
40  #####
41  $END
42
43  # solvent geometry in EFP1 (EFP2 is still not available)
44  #####3#####
45  $EFRAG
46  COORD=CART POSITION=OPTIMIZE
47  FRAGNAME=H2ODFT ! 1
48  O1      1.7760990000      4.8390610000      -2.1049530000
49  H2      0.9224740000      4.4173920000      -2.2628490000
50  H3      2.4128200000      4.1148590000      -2.1441810000
51  FRAGNAME=H2ODFT ! 2
52  O1      3.6783070000      3.8351060000      0.8717800000
53  H2      3.6020030000      4.1116030000      1.7932670000
54  H3      3.4138960000      4.6126330000      0.3648680000
55  .
56  .
57  .
58  FRAGNAME=H2ODFT ! 32
59  O1      3.7691430000      -1.4091250000      -4.0319510000
60  H2      2.8756120000      -1.5562470000      -4.3656710000
61  H3      3.6686840000      -1.3995500000      -3.0721400000

```

```
62 $END
63 #####
```

MD Input File

#####n##### are for restarting MD in case the calculation abruptly ends (see next section).

```
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=POP31 $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 C      6.0      1.1800230000      1.1968760000      -0.0402490000
30 C      6.0      1.8885400000      0.0002120000      -0.0451380000
31 C      6.0      1.1826030000      -1.1982380000      -0.0439250000
32 C      6.0      -0.2049580000      -1.2053990000      -0.0404060000
33 H      1.0      -0.7444290000      -2.1464580000      -0.0434230000
34 H      1.0      1.7166170000      -2.1417620000      -0.0464910000
35 H      1.0      2.9715860000      0.0014800000      -0.0479970000
36 H      1.0      1.7114190000      2.1418420000      -0.0377940000
37 H      1.0      -0.7497420000      2.1399280000      -0.0298190000
38 H      1.0      -2.7544730000      0.8437050000      0.2369240000
```

```

39 H          1.0          -2.7570450000          -0.8248640000          0.2993670000
40 #####
41 $END
42
43 # solvent geometry in EFP1 (EFP2 is still not available)
44 #####3#####
45 $EFRAG
46 COORD=CART POSITION=OPTIMIZE
47 FRAGNAME=H2ODFT ! 1
48 O1          1.7760990000          4.8390610000          -2.1049530000
49 H2          0.9224740000          4.4173920000          -2.2628490000
50 H3          2.4128200000          4.1148590000          -2.1441810000
51 FRAGNAME=H2ODFT ! 2
52 O1          3.6783070000          3.8351060000          0.8717800000
53 H2          3.6020030000          4.1116030000          1.7932670000
54 H3          3.4138960000          4.6126330000          0.3648680000
55 .
56 .
57 .
58 FRAGNAME=H2ODFT ! 32
59 O1          3.7691430000          -1.4091250000          -4.0319510000
60 H2          2.8756120000          -1.5562470000          -4.3656710000
61 H3          3.6686840000          -1.3995500000          -3.0721400000
62 $END
63 #####

```

TDDFT Input File

```

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 ↪
8 $SYSTEM MWORDS=200 MEMDDI=250 $END
9 #activate direct SCF calculation
10 $SCF DIRSCF=.T. $END
11 # find 5 excited states - the current setting is purely driven by its lower cost
12 # Previous experience shows that 10 states gives only a few strong peak.
13 ↪
14 $TDDFT NSTATE=5 TPA=.f. $END
15 # Basis set = 6-311++(2d,p)
16 $BASIS GBASIS=N311 NGAUSS=6 NDFUNC=2 NPFUNC=1
17 DIFFSP=.TRUE. DIFFS=.TRUE. POLAR=POPN311 $END
18 # solute geometry - C1 1 = symmetry data
19 $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 C      6.0      0.1647443916      6.2506143410      -4.1120467466
24 C      6.0      1.0257966066      6.0302470214      -5.1619968306
25 C      6.0      2.3206871944      5.6013436130      -4.8191336787
26 C      6.0      2.7816701943      5.6319869564      -3.5109071785
27 H      1.0      3.7174658997      5.1792167910      -3.3730537148
28 H      1.0      3.0442819748      5.3626012085      -5.5970784678
29 H      1.0      0.6964441939      6.0626184504      -6.2108207757
30 H      1.0      -0.8806713295      6.5494661522      -4.2694627361
31 H      1.0      -0.0805914586      6.4688905479      -2.0333995184
32 H      1.0      1.7323956480      5.6459040114      -0.4329519914
33 H      1.0      3.3564486514      5.6102990678      -1.0242941608
34 $END
35
36 # solvent geometry in EFP1 (EFP2 is still not available)
37 $EFRAG
38 COORD=CART POSITION=OPTIMIZE
39 FRAGNAME=H2ODFT ! 1
40 O1      2.335993939511      3.751856628604      1.427418842826
41 H2      1.439322965739      3.833168541986      1.710699607266
42 H3      2.854351938959      3.613582975500      2.203990690907
43 FRAGNAME=H2ODFT ! 2
44 O1      3.266753260182      2.613267395174      3.808546039622
45 H2      3.514552920456      1.768271678250      3.468757839439
46 H3      3.822697636667      2.780022240614      4.552855871622
47 .
48 .
49 .
50 FRAGNAME=H2ODFT ! 32
51 O1      -0.041331901955      -3.906598195206      1.282021099515
52 H2      -0.790424236756      -4.406259423929      1.565001283174
53 H3      -0.374908444181      -3.111491773717      0.898079798177
54 $END

```

Acknowledgement

Sed aliquam euismod nunc nec consectetur. Fusce eget dui id tortor tristique luctus. Pellentesque elit eros, molestie et molestie vitae, laoreet in risus. Nullam ligula lectus, pulvinar eget sagittis sed, cursus ac magna. ...

Supporting Information Available

Ut volutpat, felis sit amet malesuada blandit, arcu sapien feugiat libero, vel interdum ipsum dolor et dolor. Fusce tortor sapien, pharetra sit amet posuere ac, viverra mollis est. Maecenas auctor ultrices quam a pharetra. Aenean ornare dictum libero vitae gravida. Mauris auctor sapien at purus accumsan lacinia.

References

- (1) Bedoux, G.; Roig, B.; Thomas, O.; Dupont, V.; Le Bot, B. *Environmental Science and Pollution Research* **2012**, *19*, 1044–1065.
- (2) Kliegman, S.; Eustis, S. N.; Arnold, W. a.; McNeill, K. *Environmental science & technology* **2013**, *47*, 6756–63.
- (3) 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.
- (4) Lin, H.; Truhlar, D. G. *Theoretical Chemistry Accounts* **2007**, *117*, 185–199.
- (5) Cossi, M.; Barone, V. *The Journal of Chemical Physics* **2000**, *112*, 2427.
- (6) Barone, V.; Polimeno, A. *Chem. Soc. Rev.* **2007**, *36*, 1724–1731.
- (7) Li, J.; Cramer, C. J.; Truhlar, D. G. *International Journal of Quantum Chemistry* **1999**, *77*, 264–280.
- (8) Tomasi, J.; Mennucci, B.; Cammi, R. *Chemical Reviews* **2005**, *105*, 2999–3093.
- (9) Day, P. N.; Jensen, J. H.; Gordon, M. S.; Webb, S. P.; Stevens, W. J.; Krauss, M.; Garmer, D.; Basch, H.; Cohen, D. *The Journal of Chemical Physics* **1996**, *105*, 1968–1986.

- (10) Yoo, S.; Zahariev, F.; Sok, S.; Gordon, M. S. *Journal of Chemical Physics* **2008**, *129*, 1–8.
- (11) Magyar, R. J.; Tretiak, S. *Journal of Chemical Theory and Computation* **2007**, *3*, 976–987.
- (12) Defusco, A.; Minezawa, N.; Slipchenko, L. V.; Zahariev, F.; Gordon, M. S. *J. Phys. Chem. Lett* **2011**, *2*, 2184–2192.
- (13) Wiberg, K. B. *Journal of Computational Chemistry* **2004**, *25*.
- (14) Barnes, L.; Abdul-Al, S.; Allouche, A.-R. *The Journal of Physical Chemistry A* **2014**, *118*, 11033–11046.
- (15) Plugatyr, A.; Svishchev, I. M. *The Journal of chemical physics* **2009**, *130*, 114509.

This material is available free of charge via the Internet at <http://pubs.acs.org/>.