2MMN40 Report: Modeling aqueous ethanol

Bartosz Animucki (1226153),
Roman Kazus (1239966)

February 13, 2018

1 Introduction

Ever since ethyl alcohol was successfully distilled in high concentrations for the first time during the Islamic golden age [Ahm01], (al)chemists and enthusiasts alike have been fascinated by its wide uses such as a medical and industrial solvent, as a substrate in organic chemistry, as a psychoactive substance, and many more.

Nonetheless, perhaps the most important current research goal that involves the study of the properties of ethanol, especially in water mixtures, is biofuels [IK09] [VMT$^+$17]. Being derived from biomass sources, bio-ethanol contains water, no matter how efficiently it is distilled. This is because at about 95% ethanol, the mixture is azeotropic – meaning that neither ethanol nor water evaporate more when the mixture is heated. Further study of this phenomenon, as in [PAE$^+$12], would be of immense value to the feasibility of biofuel production.

The exact chemical properties of this type of water-ethanol mixtures, such as volume contraction, have also been studied extensively. One prominent theory of the origin of aqueous ethanol volume contraction, put forth by Frank and Evans [FE45], hypothesized that water molecules form so-called "icebergs" around hydrophobic solute molecules. Under this theory, when ethanol molecules are introduced to water, the water molecules around the hydrophobic end of the ethanol undergo a structural rearrangement in such a way that strong water-water hydrogen bonds are formed. This model was repeatedly tested and verified later on in prominent experimental studies such as [PB99]. Similar results were found in [GLA$^+$03].

In this report, we propose a basic computational model for predicting thermodynamic properties of water-ethanol mixtures using the principles of Molecular Modeling.

2 Theory

3 Simulation

4 Results and discussion

Verify simulation against [GLA+03]

5 Conclusion

References

[Ahm01] Ahmad Yousef al-Hassan. *Science and Technology in Islam: Technology and applied sciences*. UNESCO, 2001.

[FE45] Henry S. Frank and Marjorie W. Evans. Free volume and entropy in condensed systems iii. entropy in binary liquid mixtures; partial molal entropy in dilute solutions; structure and thermodynamics in aqueous electrolytes. *The Journal of Chemical Physics*, 13(11):507–532, 1945.

[GLA+03] J.-H. Guo, Y. Luo, A. Augustsson, S. Kashtanov, J.-E. Rubensson, D. K. Shuh, H. Ågren, and J. Nordgren. Molecular structure of alcohol-water mixtures. *Phys. Rev. Lett.*, 91:157401, Oct 2003.

[IK09] Oliver R. Inderwildi and David A. King. Quo vadis biofuels? *Energy Environ. Sci.*, 2:343–346, 2009.

[PAE+12] A.B. Pereiro, J.M.M. Araujo, J.M.S.S. Esperanca, I.M. Marrucho, and L.P.N. Rebelo. Ionic liquids in separations of azeotropic systems - A review. *The Journal of Chemical Thermodynamics*, 46:2–28, 2012. Thermodynamics of Sustainable Processes.

[PB99] Sneha A. Parke and Gordon G. Birch. Solution properties of ethanol in water. *Food Chemistry*, 67(3):241–246, 1999.

[VMT+17] Jose Luis Viviente, Jon Melendez, David Alfredo Pacheco Tanaka, Fausto Gallucci, Vincenzo Spallina, Giampaolo Manzolini, Stefano Foresti, Vincenzo Palma, Concetta Ruocco, and Leonardo Roses. Advanced m-chp fuel cell system based on a novel bio-ethanol fluidized bed membrane reformer. *International Journal of Hydrogen Energy*, 42(19):13970 – 13987, 2017. Special Issue on The 21st World Hydrogen Energy Conference (WHEC 2016), 13-16 June 2016, Zaragoza, Spain.

A Code

```
# -*- coding: utf-8 -*-
# ==========================================
# 2mmn40 code
# Water molecule simulation
# version  2018-01-16
# BA
#
#
# for BA: Make sure to run in directory
# C:\Users\20165263\Dropbox\tue\2mmn40\src
#
# ==========================================


# Objective: simulate a water molecule. So we're just
    integrating f=ma over t.
# To integrate f=ma, we need f, m, v0 and q0.
# f is obtained from potentials. f = -grad u
# m, v0, x0 are all given.

import numpy as np

# Data structure required: molecule geometry. So, a
    list of lists of molecules.
# Each molecule needs to have a mass, an x0, a v0, and
     explicit geometry

### part 1: diatomic molecule

### Parameters
#molecule parameters
bondList = [[1,2],[0],[0]]
angleList = [[1,0,2]]

#this is only for the OH bond
kbond = 1.0
rbond = 1.0

# list of masses of atoms
m = np.array([16.0, 1.0, 1.0])
```

```
#ensemble parameters
n=len(m)

#simulation parameters: choice of integrator
# 0 - forward euler
# 1 - verlet
# 2 - velocity verlet

integrator = 2
maxsteps = 2000

#output filename
output = 'outputH2O.xyz'

#initial values
q0 = np.array([[0.0, 0.0, 0.0],
               [1., 0.5, 0.0],
               [-1., 0.5, 0.0]])

v0 = np.zeros(shape=[n,3])


##END PARAMETERS

# find the bond force on a particle with index ids[0]
# as exerted by particle with index ids[1]
def findBondForces(r,dr,ids):
    pass

def findForces(qin):
    #find distance: r and dr
    dr = qin - qin[:, np.newaxis]
    r = np.linalg.norm(dr, axis=2)

    ftot = np.zeros([n,3])

    # find bond forces
    # triangular loop since forces are symmetric
    for i, bonds in enumerate(bondList):
        for j in bonds:
            if j>i:
                fij = -kbond * dr[i,j,:] * (rbond - r[
                    i,j]) / r[i,j]
                ftot[i] += fij
```

```
                ftot[j] += fij

    #implement angles
    for angle in angleList:
        pass
        #UGHHH

    return ftot

def main():
    #initialize system state
    q = [q0]
    v = [v0]

    # take a small enough timestep
    dt = min(np.sqrt( kbond/m )) /100

    #integrator selection
    if integrator==0:
        #Forward Euler
        for i in range(maxsteps):
            f = findForces(q[i])
            #integration step
            q.append(q[i] + dt*v[i] + dt**2 /(2*m[:,np
                .newaxis]) *f)
            v.append(v[i] + dt/m[:,np.newaxis] *f)

    elif integrator==1:
        #Verlet
        # First step: make an Euler step on q
        # we do this because Verlet needs two q states
            in the past
        f=findForces(q[0])
        q.append(q[0] + dt*v[0] + dt**2 /(2*m[:,np.
            newaxis]) *f)

        for i in range(1,maxsteps):
            f = findForces(q[i])
            q.append(2*q[i] - q[i-1] + dt**2 / m[:, np
                .newaxis] *f)
            v.append((q[i+1]-q[i-1])/2)

    elif integrator==2:
        #Velocity Verlet
```

```
        f = findForces(q[0])
        for i in range(maxsteps):
            q.append(q[i] + dt*v[i] + dt**2 / (2*m[:,
                np.newaxis]) *f)
            # here the forces have to be known at time
                t and at time t+dt;
            # so we find the forces now, and use this
                in the next run
            fnext = findForces(q[i+1])
            v.append(v[i] + dt/(2*m[:, np.newaxis]) *
                (f + fnext))
            f = fnext

    else:
        raise Exception('Invalid integrator {}'.format
            (integrator))

    #export result to xyz
    with open(output,'w') as outfile:
        #enumerate iterates through a list, and lets
            use an index too
        for i, snapshot in enumerate(q):
            print('{}'.format(n), file=outfile)
            print('Water molecule. t = {:10.5f}'.
                format(i*dt),
                    file=outfile)
            #create list of triples: first dimension
                is number of molecules
            molecules = snapshot.reshape([n//3,3,3])
            for molecule in molecules:
                #slightly hacky starred expression in
                    format: unpacking
                print('O {:10.5f} {:10.5f} {:10.5f}'
                        .format(*molecule[0],),
                        file=outfile)
                print('H {:10.5f} {:10.5f} {:10.5f}'
                        .format(*molecule[1],),
                        file=outfile)
                print('H {:10.5f} {:10.5f} {:10.5f}'
                        .format(*molecule[2],),
                        file=outfile)

#for running script as file:
#run main
```

main ( )