NUEN 647
Uncertainty Quantification for Nuclear Engineering
Assignment 1

Due on Tuesday, October 4, 2016

Dr. McClarren

Paul Mendoza

Contents

Complete the exercises in the Chapter 2 notes. Be sure to include discussion of results where appropriate. You may use any tools that are approrpriate to solving the problem.

Problem 1

Listing 1 shows a Perl script.

Listing 1: Sample Perl Script With Highlighting

```perl
#!/usr/bin/perl

use strict;
use warnings;

for (1..99) { print $_." Luftballons\n"; }

# This is a commented line

my $string = "Hello World!";

print $string."\n\n";

$string =~ s/Hello/Goodbye/;

print $string."\n\n";

test();

exit;

sub test { print "All good.\n"; }
```

Listing 2: Sample python script no .py

```python
#!/usr/bin/env python3

"""
Chem Calculations
"""

__author__     =  "Paul Mendoza"
__copyright__  =  "Copyright 2016, Planet Earth"
__credits__    = ["Sunil Chirayath",
                  "Charles Folden",
                  "Jeremy Conlin"]
__license__    =  "GPL"
__version__    =  "1.0.1"
__maintainer__ =  "Paul Mendoza"
__email__      =  "paul.m.mendoza@gmail.com"
__status__     =  "Production"

################################################################
#################### Import packages ##########################
```

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

import os.path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from uncertainties import ufloat
from uncertainties.umath import *
from uncertainties import unumpy as unp
import re
import time
start_time = time.time()

import Functions as fun


################################################################
################## Examples of Calculations ###################
################################################################

#######################################################
############ Atom Fraction to Mass Fraction ##########
############          and vice versa        ##########
#######################################################

string='92235 0.285714286 0 92238 0.714285714 0'
MasstoAtom=True
Mass,Zaid=fun.StringToMass(string)
stringCalculated=fun.ConvertFractions(string,Mass,MasstoAtom,Zaid)

# if MasstoAtom:
#     print("Mass Fractions:")
#     print(string)
#     print("Atom Fractions:")
#     print(stringCalculated)
# else:
#     print("Mass Fractions:")
#     print(stringCalculated)
#     print("Atom Fractions:")
#     print(string)

#######################################################
############ Calculate grams per mol of ##############
############    a chemical formula      ##############
#######################################################

#Make sure your chemical form has no repeats
#And no parentheses
ChemicalFormula='HNO_3'
ChemicalFormulaError=[0,0,0,0,0,0,0] #+/- error in integers of
                                     #chemical formula
ChemicalFormula=ChemicalFormula+"    "
```

```
List=fun.ChemList(ChemicalFormula)

#Enter Modifications:
#1. Each element should be a single item in the list
#2. Format: zaid atomfraction+/-error zaid atomfraction+/-error
#   or    : zaid atomfraction error zaid atomfraction
Modifications=['92235 0.2883155436+/-0.0000000024 92238 0.7116844564+/-0.0000000024',stringCalcu

df = pd.read_csv('../Data/AtomicWeights.csv')


ModMass,ModSymbols,AtomFractions=fun.FormatMods(Modifications,df)
MolarMass=fun.DetermineMolarMass(List,df,
                                 ModSymbols,ModMass,
                                 AtomFractions,ChemicalFormulaError)

# print(MolarMass)

#######################################################
############### Calculate Molality from ###############
###############          Wt %           ###############
#######################################################

gramsOmol=MolarMass
WtConcentration=ufloat(69,0.1)
Molality=1000/(gramsOmol*(100/WtConcentration-1))

#######################################################
############# Convert molality/molarity ###############
#######################################################

MolarityToMolality=True

gramsOmol=MolarMass

#Density in grams per cc or grams per ml
dfDen=pd.read_csv('../Data/Nitric_Acid.csv')

Temperature=ufloat(20,3) #Same degrees as dfDen!!!

Molality=Molality
Molarity=ufloat(15.43,0.06)

if MolarityToMolality:
    Molality=fun.ConvertMol(MolarityToMolality,Molarity,
                            gramsOmol,dfDen,Temperature)
else:
    Molarity=fun.ConvertMol(MolarityToMolality,Molality,
                            gramsOmol,dfDen,Temperature)



#print("Molarity = "+str(Molarity))
#print("Molality = "+str(Molality))
```

```


########################################################
############# Calculate New Concentration ############
########################################################

Vol1=1
Vol2=1

gramsOmol=gramsOmol

m1=Molality
m2=Molality*.25

Temperature=ufloat(20,3)

dfDen=pd.read_csv('../Data/Nitric_Acid.csv')

m3,p3,Vol3,Wt,M3=fun.NewConcentration(m1,m2,gramsOmol,
                                Temperature,dfDen,
                                Vol1,Vol2)

M1=fun.ConvertMol(False,m1,gramsOmol,dfDen,Temperature)
M2=fun.ConvertMol(False,m2,gramsOmol,dfDen,Temperature)


###################### Time To execute #################

print("--- %s seconds ---" % (time.time() - start_time))
```

Listing 3: Sample python script no .py

```
#!/usr/bin/env python3

"""
FractionAM converts atom fractions to mass fractions
and mass fractions to atom fractions. Input is a
single string with MCNP style fractions.
"""

__author__     =  "Paul Mendoza"
__copyright__  =  "Copyright 2016, Planet Earth"
__credits__    = ["Sunil Chirayath",
                  "Charles Folden",
                  "Jeremy Conlin"]
__license__    =  "GPL"
__version__    =  "1.0.1"
__maintainer__ =  "Paul Mendoza"
__email__      =  "paul.m.mendoza@gmail.com"
__status__     =  "Production"

################################################################
#################### Import packages #########################
################################################################
```

```python
import os.path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from uncertainties import ufloat
from uncertainties.umath import *
from uncertainties import unumpy as unp
import re

################################################################
######################## Functions ###########################
################################################################

def ReturnUfloat(string):
    """
    string has format   238.023249814(23)
            or format   [15.99903-15.99977]
            or format   235.04+/-0.0000019

    Returns a uncertain number so python can do calculations
    """
    if "(" in string:
        Number=str(string.split('(')[0])
        LastErrorNumber=str(string.split("(")[1].replace(")",""))
        NumberOfZeros=len(Number.split(".")[1])-len(LastErrorNumber)
        Error="0."
        for i in range(0,NumberOfZeros):
            Error=Error+"0"
        Error=Error+LastErrorNumber
    elif "[" in string:
        FirstNum=float(string.split('-')[0].replace("[",''))
        SecondNum=float(string.split('-')[1].replace(']',''))
        Number=str((FirstNum+SecondNum)/2)
        Error=str(float(Number)-FirstNum)
    elif "+/-" in string:
        Number=string.split("+/-")[0]
        Error=string.split("+/-")[1]

    return(ufloat(float(Number),float(Error)))

def FindAtomicMass(df,proton,Isotope):
    """
    This function will take in a dataset 'df' look through the
    'df.Protons' column and find the column that matches with
    'proton'. If the row that contains 'proton' also contains
    'Isotope' in the 'df.Isotope' column, then the value stored
    in 'df.Relative_Atomic_Mass' is reported for that row.
    Because the proton numbering scheme can have a format
    '10' for hydrogen and '10' for neon (following MCNP ZAID
    naming conventions) if we don't find a value with the whole
    string of 'proton' then the program looks through the first
    element of string and tries to match that 'proton[0]'
    If no matches are found, and error is thrown out.

    df = dataset with columns 'Protons' 'Isotopes' and
    'Relative_Atomic_Mass'. Dataset created with pandas

    proton = string with proton number (follow MCNP zaid format)

    Isotope = string with isotope number (just put the atomic mass
    do not follow MCNP format - different for few cases)
    """
    #print(df)
    for i in range(0,len(df.Protons)):
        dfPro=str(df.Protons[i])
        if proton==dfPro:
            dfIso=str(df.Isotope[i])
            if Isotope==dfIso:
                Mass=df.Relative_Atomic_Mass[i]
                break
    try:
        Mass
    except NameError:
        for i in range(0,len(df.Protons)):
            dfPro=str(df.Protons[i])
            if proton[0]==dfPro:
                dfIso=str(df.Isotope[i])
                if Isotope==dfIso:
                    Mass=df.Relative_Atomic_Mass[i]
                    break
    try:
        Mass
    except NameError:
        print("Could not find atomic mass for proton = "\
              +proton+" and for Isotope = "+Isotope)
    Mass=ReturnUfloat(Mass)
    return(Mass)

def CheckInParen(i,ChemicalFormula):
    """
    i = index inside the string 'ChemicalFormula
    ChemicalFormula = string that could potentially have ()

    Please note, this code is not complete
    """
    NumberOpenParen=ChemicalFormula.count("(")
    NumberCloseParen=ChemicalFormula.count(")")
    if NumberOpenParen != NumberCloseParen:
        print("Unbalanced parentheses in chemical formula")
        quit()
    if NumberOpenParen==0:
        return(1,False)

    print("Hello")
```

Problem 1 continued on next page. . .

```python
    Mul=4
    Test=True
    return(Mul,Test)

def ChemList(ChemicalFormula):
    """
    This function will take in a string for a
    chemical formula.

    Please modify your formula to fit the following rules

    1. No repeats of elements (sum up all the same time element)
    2. To enter a subscript use "_", for example He_3 indicates
       three helium atoms.
    3. Use captical letters for the first letter of an element.
       If there are multiple letters for an elemental symbol,
       then use lowercase for the second letter (program does
       not interpret three symbol elements)
    4. If there are more than 999 of a single atom in your chemical
       formula, you will have to write your own code. Or modify
       this one.
    """
    i=0
    List=[]
    while (i <len(ChemicalFormula)-1):
        start=i
        #print("The beginning i index = "+str(i))
        if re.search('[A-Z]',ChemicalFormula[i]):                #Capital letter?
            if re.search('[a-z]',ChemicalFormula[i+1]):          #Followed by lowercase?
                if re.search('_',ChemicalFormula[i+2]):          #Followed by more than 1?
                    if re.search('[0-9]',ChemicalFormula[i+5]): #Hundreds check
                        List=np.append(List,ChemicalFormula[i:i+6])
                        #print(ChemicalFormula[i:i+6])
                        i=i+6
                    elif re.search('[0-9]',ChemicalFormula[i+4]): #tens check
                        List=np.append(List,ChemicalFormula[i:i+5])
                        #print(ChemicalFormula[i:i+5])
                        i=i+5
                    else:                                         #If not hundres or tens, then c
                        List=np.append(List,ChemicalFormula[i:i+4])
                        #print(ChemicalFormula[i:i+4])
                        i=i+4
                else:                                            #If not more than one, print
                    List=np.append(List,ChemicalFormula[i:i+2])
                    #print(ChemicalFormula[i:i+2])
                    i=i+2
            elif re.search('_',ChemicalFormula[i+1]):            #If only single symbol, then do
                if re.search('[0-9]',ChemicalFormula[i+4]):      #hundreds
                    List=np.append(List,ChemicalFormula[i:i+5])
                    #print(ChemicalFormula[i:i+5])
                    i=i+5
                elif re.search('[0-9]',ChemicalFormula[i+3]):    #tens
                    List=np.append(List,ChemicalFormula[i:i+4])
```

```
                        #print(ChemicalFormula[i:i+4])
                        i=i+4
                    else:                                               #ones
                        List=np.append(List,ChemicalFormula[i:i+3])
                        #print(ChemicalFormula[i:i+3])
                        i=i+3
                else:
                    List=np.append(List,ChemicalFormula[i])
                    #print(ChemicalFormula[i])
                    i=i+1
        if start==i: #If we didn't find anything useful
            i=i+1
        #print("The end i index = "+str(i))
    return(List)

def StringToMass(string):
    """
    This function takes in a string of the form
    zaid fraction error zaid fraction error ...
    will read a file called 'AtomicWeights.csv'
    and find the atomic weight with error of the zaids
    and store those value in a list called Mass
    """
    ListOfString=string.split()

    if not len(ListOfString)%3==0:
        print("Check string variable missing fraction or error")
        quit()

    #Initialize fractions and zaid
    Zaid=0*np.arange(0,int(len(ListOfString)/3))

    #Gather fraction data and zaid data
    for i in range(0,int(len(ListOfString)/3)):
        Zaid[i]=int(ListOfString[i*3])


    df = pd.read_csv('../Data/AtomicWeights.csv')
    #Gather Mass Data
    for i in range(0,len(Zaid)):
        sZaid=str(Zaid[i])
        if len(sZaid)==4:
            proton=sZaid[0:2]
            if sZaid[2]=="0":
                Isotope=sZaid[3]
            else:
                Isotope=sZaid[2:4]
        elif len(sZaid)==5:
            proton=sZaid[0:2]
            if sZaid[2]=="0":
                Isotope=sZaid[3:5]
            if sZaid[3]=="0":
                Isotope=sZaid[4:5]
            if sZaid[2]!="0" and sZaid[3]!="0":
                Isotope=sZaid[2:5]
        elif len(sZaid)==6:
            proton=sZaid[0:3]
            Isotope=sZaid[3:6]
        else:
            print("Length of zaid is not 4 5 or 6 err")
            quit()
        try:
            Mass=np.append(Mass,FindAtomicMass(df,proton,Isotope))
        except NameError:
            Mass=FindAtomicMass(df,proton,Isotope)

    return(Mass,Zaid)

def StringToMass2(string):
    """
    This function takes in a string of the form
    zaid fraction error zaid fraction error ...
    will read a file called 'AtomicWeights.csv'
    and find the atomic weight with error of the zaids
    and store those value in a list called Mass
    """
    ListOfString=string.split()

    if not len(ListOfString)%3==0:
        print("Check string variable missing fraction or error")
        quit()

    #Initialize fractions and zaid
    Zaid=0*np.arange(0,int(len(ListOfString)/3))

    #Gather fraction data and zaid data
    for i in range(0,int(len(ListOfString)/3)):
        Zaid[i]=int(ListOfString[i*3])
        floatednumber=ufloat(float(ListOfString[i*3+1]),
                             float(ListOfString[i*3+2]))
        try:
            AtomFractions=np.append(AtomFractions,floatednumber)
        except NameError:
            AtomFractions=floatednumber

    df = pd.read_csv('../Data/AtomicWeights.csv')
    #Gather Mass Data
    for i in range(0,len(Zaid)):
        sZaid=str(Zaid[i])
        if len(sZaid)==4:
            proton=sZaid[0:2]
            if sZaid[2]=="0":
                Isotope=sZaid[3]
            else:
                Isotope=sZaid[2:4]
        elif len(sZaid)==5:
            proton=sZaid[0:2]
            if sZaid[2]=="0":
                Isotope=sZaid[3:5]
            if sZaid[3]=="0":
                Isotope=sZaid[4:5]
            if sZaid[2]!="0" and sZaid[3]!="0":
                Isotope=sZaid[2:5]
        elif len(sZaid)==6:
            proton=sZaid[0:3]
            Isotope=sZaid[3:6]
        else:
            print("Length of zaid is not 4 5 or 6 err")
            quit()
        try:
            Mass=np.append(Mass,FindAtomicMass(df,proton,Isotope))
            protons=np.append(protons,proton)
        except NameError:
            Mass=FindAtomicMass(df,proton,Isotope)
            protons=proton

    return(Mass,protons,AtomFractions)

def ConvertFractions(string,Mass,MasstoAtom,Zaid):
    """
    This function will convert, with error, the mass or atom fraction
    to the other (mass to atom or atom to mass). It will use the masses
    provided in Mass, and the fractions provided in string. If its mass to Atom then
    MasstoAtom=True, otherwise set False
    """

    ListOfString=string.split()
    Total=ufloat(0.,0)

    for i in range(0,len(Zaid)):

        Fraction=ufloat(float(ListOfString[i*3+1]),float(ListOfString[i*3+2]))
        if MasstoAtom: #Calculate total Atoms
            Total=Total+Fraction/Mass[i]
        else: #Calculate total Mass
            Total=Total+Fraction*Mass[i]

    stringCalculated=''
    for i in range(0,len(Zaid)):

        Fraction=ufloat(float(ListOfString[i*3+1]),float(ListOfString[i*3+2]))
        if MasstoAtom:
            #Calculate atom fractions
            FractionCalculated=(Fraction/Mass[i])/Total
        else:
            #Calculate mass fractions
            FractionCalculated=(Fraction*Mass[i])/Total

        stringCalculated=stringCalculated+\
```

```python
                              str(Zaid[i])+' '+\
                              str(FractionCalculated)+' '

    return(stringCalculated)


def FindSymbol(NumofProtons,df):
    """
    This function will find the element symbol, based on number of
    protons.
    """
    for i in range(0,len(df.Protons)):
        if str(df.Protons[i])==NumofProtons:
            Symbol=df.Symbol[i]
            break

    try:
        Symbol
    except NameError:
        print("Could not find Symbol for Modfication zaid")
        quit()

    return(Symbol)

def FormatMods(Modifications,df):
    """
    This functions formats modifications

    """
    for i in range(0,len(Modifications)):
        Modifications[i]=Modifications[i].replace('+/-',' ')

        Mass,protons,AtomFractions=StringToMass2(Modifications[i])
        Mass=" ".join(str(i) for i in Mass)
        protons=" ".join(str(i) for i in protons)
        LAtomFractions=" ".join(str(i) for i in AtomFractions)
        try:
            ModMass=np.append(ModMass,Mass)
            Modprotons=np.append(Modprotons,protons)
            ModAFrac=np.append(ModAFrac,LAtomFractions)
        except NameError:
            ModMass=[Mass]
            Modprotons=[protons]
            ModAFrac=[LAtomFractions]


    for i in range(0,len(Modifications)):
        proton=Modprotons[i].split(" ")[0]
        symbol=FindSymbol(proton,df)
        try:
            ModSymbols=np.append(ModSymbols,symbol)
        except NameError:
            ModSymbols=symbol
    return(ModMass,ModSymbols,ModAFrac)

def DetermineMolarMass(List,df,ModSymbols,
                       ModMass,AtomFractions,
                       ChemicalFormulaError):
    """
    this function determines the molar mass of a chemical formula
    with error:
    List is a list of the chemical formula
    df is a dataframe with atomic mass information
    ModSymbols are the modificaiton symbols (if using different Dudes
    AtomFractions are the atom fractions of the different dudes
    ChemicalFormulaError is the error in the number of each atom in the
    chemical formula, for example UO_2 could have a chemical formula
    ChemicalFormulaError=[0,0.001], meaning that a very small amount of
    the time, we have UO_3...this isn't the best way of doing this...
    """
    MolarMass=ufloat(0,0)
    for i in range(0,len(List)):
        Symbol=List[i].split("_")[0]
        try:
            Multiplier=List[i].split("_")[1]
        except IndexError:
            Multiplier=1
        Multiplier=ufloat(Multiplier,ChemicalFormulaError[i])
        for j in range(0,len(df.Symbol)):
            if Symbol==str(df.Symbol[j]):
                ModifyElement=False
                for k in range(0,len(ModSymbols)):
                    if ModSymbols[k]==Symbol: #We are modifying
                        ModifyElement=True
                        Masses=ModMass[k].split(" ")
                        AFractions=AtomFractions[k].split(" ")
                        IndividualMolarMass=0
                        for l in range(0,len(Masses)):
                            IndividualMolarMass=IndividualMolarMass+\
                                        ReturnUfloat(Masses[l])*\
                                        ReturnUfloat(AFractions[l])
                if not ModifyElement:
                    IndividualMolarMass=ReturnUfloat(
                                            df.Standard_Atomic_Weight[j]
                                                    )
                # print(Symbol+" "+
                #       str(IndividualMolarMass)
                #      )
                MolarMass=MolarMass+IndividualMolarMass*Multiplier
                break
    return(MolarMass)

def FindRange(List,Item):
    """
    This function returns a range...yup
    """
    for i in range(0,len(List)-1):
        if List[i] == Item:
            Range=[List[i]]
            break
        elif List[i+1] == Item:
            Range=[List[i+1]]
            break
        elif List[i] <= Item <= List[i+1]:
            Range=[List[i],List[i+1]]
            break
    return(Range)

def FindInTable(List1,List2,ItemMatchWithList2):
    """
    This function needs two lists that are the same
    length. and with data that corresponds to each other
    searches through list2 to find the item,
    then reports that same value from list1
    """
    for i in range(0,len(List2)):
        if(ItemMatchWithList2==List2[i]):
            return(List1[i])

def InterpolateDensity(dfDen,Temp,TRange,Conc,CRange):
    """
    This function interpolates stuff...don't ask me how
    """
    Concentrations=dfDen['Concentration_Percent_Weight']

    for i in range(0,len(TRange)):
        t=dfDen[str(int(TRange[i]))+'  C']
        for j in range(0,len(CRange)):
            C=CRange[j]
            D=FindInTable(t,Concentrations,C)
            try:
                Densities=np.append(Densities,D)
            except NameError:
                Densities=[D]

    if len(Densities)==4:
        Q11=((TRange[1]-Temp)*(CRange[1]-Conc))/\
            ((TRange[1]-TRange[0])*(CRange[1]-CRange[0]))
        Q21=((Temp-TRange[0])*(CRange[1]-Conc))/\
            ((TRange[1]-TRange[0])*(CRange[1]-CRange[0]))
        Q12=((TRange[1]-Temp)*(Conc-CRange[0]))/\
            ((TRange[1]-TRange[0])*(CRange[1]-CRange[0]))
        Q22=((Temp-TRange[0])*(Conc-CRange[0]))/\
            ((TRange[1]-TRange[0])*(CRange[1]-CRange[0]))

        density=Q11*Densities[0]+Q12*Densities[1]+\
                Q21*Densities[2]+Q22*Densities[3]
    if len(Densities)==1:
        density=Densities[0]

    if len(Densities)==2:
        if len(TRange)==2:
            density=((Temp-TRange[0])*(Densities[1]-Densities[0]))/\
                    (TRange[1]-TRange[0])+Densities[0]
        if len(CRange)==2:
            density=((Conc-CRange[0])*(Densities[1]-Densities[0]))/\
                    (CRange[1]-CRange[0])+Densities[0]

    #print(density)
    #print(Temp)
    #print(Conc)
    return(density)

def GetDensity(Temperature,WtConcentration,dfDen):
    """
    This function gets you density, don't ask me how
    """
    MinTemp=Temperature.nominal_value-Temperature.std_dev
    MaxTemp=Temperature.nominal_value+Temperature.std_dev
    MinWtCon=WtConcentration.nominal_value-WtConcentration.std_dev
    MaxWtCon=WtConcentration.nominal_value+WtConcentration.std_dev

    Columns=list(dfDen.columns.values)

    #Find all the temperatures
    for i in range(0,len(Columns)):
        if (' C ' in Columns[i]):
            Temp=float(Columns[i].split(" C ")[0])
            try:
                TempsAva=np.append(TempsAva,Temp)
            except NameError:
                TempsAva=Temp

    #Find the temperatures you fit between
    MinTempRange=FindRange(TempsAva,MinTemp)
    MaxTempRange=FindRange(TempsAva,MaxTemp)

    #Find all the concentrations
    for i in range(0,len(dfDen.Concentration_Percent_Weight)):
        StrCon=float(dfDen.Concentration_Percent_Weight[i])
        try:
            Concentration=np.append(Concentration,StrCon)
        except NameError:
            Concentration=StrCon

    #Find concentrations you fit between
    MinConRange=FindRange(Concentration,MinWtCon)
    MaxConRange=FindRange(Concentration,MaxWtCon)

    density=InterpolateDensity(dfDen,
```

Problem 1 continued on next page. . .

```
                                MinTemp,
                                MinTempRange,
                                MinWtCon,
                                MinConRange)

    density=np.append(density,InterpolateDensity(dfDen,
                                                  MinTemp,
                                                  MinTempRange,
                                                  MaxWtCon,
                                                  MaxConRange))

    density=np.append(density,InterpolateDensity(dfDen,
                                                  MaxTemp,
                                                  MaxTempRange,
                                                  MinWtCon,
                                                  MinConRange))

    density=np.append(density,InterpolateDensity(dfDen,
                                                  MaxTemp,
                                                  MaxTempRange,
                                                  MaxWtCon,
                                                  MaxConRange))

    densityN=(max(density)+min(density))/2
    densityE=densityN-min(density)
    density=ufloat(densityN,densityE)
    return(density)

def ConvertMol(MolarityToMolality,First,
           gramsOmol,dfDen,Temperature):
    """
    This function will convert molality to molarity
    and viceversa
    """
    #First either equals Molarity or Molality
    #Second either equals Molarity or Molality
    if not MolarityToMolality:
        WtConcentration=100/(1000/(First*gramsOmol)+1)
    else:
        dif=1
        WtConcentration=ufloat(30,0.1) #A Guess
        while( abs(dif)>0.001):
            OldWt=WtConcentration
            density=GetDensity(Temperature,OldWt,dfDen)
            WtConcentration=(100*gramsOmol*First)/(1000*density)
            dif=(WtConcentration-OldWt)/WtConcentration

    density=GetDensity(Temperature,WtConcentration,dfDen)

    ##################################################
    ################## Calculation ###################
    ##################################################

```

Problem 1 continued on next page. . .

```
    if MolarityToMolality:
        #(mols/kg)
        #dif=1
        #while (abs(dif)>0.001):
            #NewSecond=1/(density/First-gramsOmol*0.001)
            #WtConcentration=100/(1000/(NewSecond*gramsOmol)+1)
            #density=GetDensity(Temperature,WtConcentration,dfDen)
            #Second=1/(density/First-gramsOmol*0.001)
            #dif=Second-NewSecond
        Second=1/(density/First-gramsOmol*0.001)
    else:
        #(mols/L)
        Second=density/(1/First+gramsOmol*0.001)

    return(Second)

def NewConcentration(m1,m2,gramsOmol,
                     Temperature,dfDen,
                     Vol1,Vol2):
    """
    This function calculates a new concentration when
    two volumes of the same substance are added together
    same temperature, assuming that both solutions
    have had time to cool
    """

    WtConcentration1=100/(1000/(m1*gramsOmol)+1)
    WtConcentration2=100/(1000/(m2*gramsOmol)+1)

    p1=GetDensity(Temperature,WtConcentration1,dfDen)
    p2=GetDensity(Temperature,WtConcentration2,dfDen)

    molsV1=(m1*gramsOmol*p1*Vol1)/(1000*gramsOmol+m1*(gramsOmol**2))
    molsV2=(m2*gramsOmol*p2*Vol2)/(1000*gramsOmol+m2*(gramsOmol**2))

    #kgSol1=(1000*p1*Vol1)/(1000+m1*gramsOmol)/1000
    #kgSol2=(1000*p2*Vol2)/(1000+m2*gramsOmol)/1000

    kgSol1=(1-WtConcentration1/100)*(p1*Vol1)/(1000)
    kgSol2=(1-WtConcentration2/100)*(p2*Vol2)/(1000)

    Totmols=molsV1+molsV2
    Totkg=kgSol1+kgSol2

    m3=Totmols/Totkg
    WtConcentration3=100/(1000/(m3*gramsOmol)+1)
    #Assuming its had time to cool down
    p3=GetDensity(Temperature,WtConcentration3,dfDen)
    Vol3=(p1*Vol1+p2*Vol2)/p3

    Molarity=ConvertMol(False,m3,gramsOmol,dfDen,Temperature)

    return(m3,p3,Vol3,WtConcentration3,Molarity)
```

Problem 1 continued on next page. . .

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Problem 2

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

This is an example citation [1].

References

[1] E. T. Tatro, S. Hefler, S. Shumaker-Armstrong, B. Soontornniyomkij, M. Yang, A. Yermanos, N. Wren, D. J. Moore, and C. L. Achim. Modulation of bk channel by microrna-9 in neurons after exposure to hiv and methamphetamine. *J Neuroimmune Pharmacol*, 2013. Tatro, Erick T Hefler, Shannon Shumaker-Armstrong, Stephanie Soontornniyomkij, Benchawanna Yang, Michael Yermanos, Alex Wren, Nina Moore, David J Achim, Cristian L R03 DA031591/DA/NIDA NIH HHS/United States U19 AI096113/AI/NIAID NIH HHS/United States Journal article Journal of neuroimmune pharmacology : the official journal of the Society on NeuroImmune Pharmacology J Neuroimmune Pharmacol. 2013 Mar 19.