```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 18 22:41:20 2020

@author: maxhu
collaborator grant
"""

#============================================================================#
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
plt.style.use('classic')
#============================================================================#
SpecificHeatofHCl = .99894
massofHCl = 100
SmoothFactor = 10
width = 2
MassofTRIS = .499
MMofGlycine = 75.07
CalorietoJConversion = 4.184
T = 298.15
R = 8.314
#============================================================================#
def DataExtractor(x,y,path):
    lines = np.loadtxt(path)
    for line in lines:
        x.append(line[0])
        y.append(line[1])

def LinearRegression(x,y):
    m, b, r_value, p_value, std_err = stats.linregress(
            x,y)
    return m*x + b

def TemperatureChange(TempData, Solution):
    ChangeInTemp = round(np.max(TempData) - np.min(TempData), 2)
    if TempData[0] < TempData[len(TempData)-1-SmoothFactor]:
        T_63 = TempData[len(TempData) - SmoothFactor] - (1 - .63) *
        abs(ChangeInTemp)
        plt.axhline(T_63, alpha=.7, linestyle='dotted', color='black',
                    label='$T_{.63}$')
        plt.legend(loc='best')
        print('The change in temperature of {} is {}K and T_63 is {}K'.format(
                Solution, ChangeInTemp, round(T_63,2)))
    elif TempData[0] > TempData[len(TempData)-SmoothFactor]:
        T_63 = (1 - .63) * abs(ChangeInTemp) + TempData[len(TempData)-
               SmoothFactor]
        plt.axhline(T_63, alpha=.9, linestyle='dotted', color='black',
                    label='$T_{.63}$')
        plt.legend(loc='best')
        print('The change in temperature of {} is {}K and T_63 is {}K'.format(
                Solution, ChangeInTemp, round(T_63,2)))
    return ChangeInTemp
```

```
def Smooth(y, N):
    cumsum = np.cumsum(np.insert(y, 0, 0))
    y_smooth = (cumsum[N:] - cumsum[:-N]) / float(N)
    return y_smooth

def Enthalpy(TempData, MassofGlycine, Solution):
    ChangeInTemp = round(np.min(TempData) - np.max(TempData), 2)
    Q_n = -HeatCapacityofCalorimeter * ChangeInTemp
    H = Q_n / (MassofGlycine / MMofGlycine)
    if Solution == 'NULL':
        pass
    else:
        print('The enthalpy of {} is {}kJ'.format(Solution, round(H * 1e-3,2)))
    return H

def ErrorInEnthalpy(Enthalpy, Reaction, ExpectedEnthalpy):
    Error = abs(((ExpectedEnthalpy - Enthalpy) / ExpectedEnthalpy)*100)
    print("The %err for {}'s enthalpy is {}%".format(Reaction, round(Error,1)))
#============================================================================#
StandardTime = []
StandardTemp = []
#============================================#
Solution1Time = []
Solution1Temp = []
#============================================#
Solution2Time = []
Solution2Temp = []
#============================================#
Solution3Time = []
Solution3Temp = []
#============================================================================#
DataExtractor(StandardTime,StandardTemp,'Cleaned_Data\standard.txt')
StandardTime = np.array(StandardTime)
StandardTemp = np.array(StandardTemp) + 273.15
#============================================#
DataExtractor(Solution1Time,Solution1Temp,'Cleaned_Data\sol_1.txt')
Solution1Time = np.array(Solution1Time)
Solution1Temp = np.array(Solution1Temp) + 273.15
#============================================#
DataExtractor(Solution2Time,Solution2Temp,'Cleaned_Data\sol_2.txt')
Solution2Time = np.array(Solution2Time)
Solution2Temp = np.array(Solution2Temp) + 273.15
#============================================#
DataExtractor(Solution3Time,Solution3Temp,'Cleaned_Data\sol_3.txt')
Solution3Time = np.array(Solution3Time)
Solution3Temp = np.array(Solution3Temp) + 273.15
#============================================================================#
size = 20
size_config = .8
fig = plt.figure(1,figsize=(15,10))
fig.suptitle('$Temperature\ as\ a\ Function\ of\ Time$', fontsize=size)
plt.tick_params(labelcolor='none', top='off', bottom='off', left='off',
                right='off')
plt.ticklabel_format(style='plain', axis='both', scilimits=(0,0))
plt.xlabel('$Time\ (s)$', fontsize=size_config*size)
plt.ylabel('$Temperature\ (K)$', fontsize=size_config*size)
```

```
#=========================================#
fig.add_subplot(221)
plt.plot(StandardTime,StandardTemp, color='black', label='$Standard$')
xDataSmoothed = np.linspace(0,120,len(Smooth(StandardTemp, SmoothFactor)))
plt.plot(xDataSmoothed,Smooth(StandardTemp, SmoothFactor),
         label='$Standard\ Smoothed$', linewidth=width)
plt.legend(loc='best')
TemperatureChange(Smooth(StandardTemp, SmoothFactor), 'standard')

fig.add_subplot(222)
plt.plot(Solution1Time,Solution1Temp, color='black', label='$Solution\ 1$')
xDataSmoothed = np.linspace(0,120,len(Smooth(Solution1Temp, SmoothFactor)))
plt.plot(xDataSmoothed,Smooth(Solution1Temp, SmoothFactor),
         label='$Solution\ 1\ Smoothed$', linewidth=width)
plt.legend(loc='best')
TemperatureChange(Smooth(Solution1Temp, SmoothFactor), 'solution 1')

fig.add_subplot(223)
plt.plot(Solution2Time,Solution2Temp, color='black', label='$Solution\ 2$')
xDataSmoothed = np.linspace(0,120,len(Smooth(Solution2Temp, SmoothFactor)))
plt.plot(xDataSmoothed,Smooth(Solution2Temp, SmoothFactor),
         label='$Solution\ 2\ Smoothed$', linewidth=width)
plt.legend(loc='best')
TemperatureChange(Smooth(Solution2Temp, SmoothFactor), 'solution 2')

fig.add_subplot(224)
plt.plot(Solution3Time,Solution3Temp, color='black', label='$Solution\ 3$')
xDataSmoothed = np.linspace(0,120,len(Smooth(Solution3Temp, SmoothFactor)))
plt.plot(xDataSmoothed,Smooth(Solution3Temp, SmoothFactor),
         label='$Solution\ 3\ Smoothed$', linewidth=width)
plt.legend(loc='best')
TemperatureChange(Smooth(Solution3Temp, SmoothFactor), 'solution 3')
#=========================================#
plt.savefig('Solution_Calorimetry.eps')
#=============================================================================#
StandardTemp = Smooth(StandardTemp, SmoothFactor)
ChangeInTempofStandard = np.max(StandardTemp) - np.min(StandardTemp)
T_63 = StandardTemp[len(StandardTemp) - SmoothFactor] - (1 - .63) *
abs(ChangeInTempofStandard) - 273.15
HeatCapacityofHCl = massofHCl * SpecificHeatofHCl
HeatofStandard = MassofTRIS * (58.738 + 0.3433 * (25 - T_63))
TotalHeatCapacity = HeatofStandard / ChangeInTempofStandard
HeatCapacityofCalorimeter = CalorietoJConversion * (TotalHeatCapacity -
                                                    HeatCapacityofHCl)

EnthalpyforReaction1 = Enthalpy(Solution1Temp, 1.5035, 'reaction 1')
EnthalpyforReaction1
EnthalpyforReaction2 = Enthalpy(Solution3Temp, 1.5048, 'NULL') -
Enthalpy(Solution2Temp, 1.5016, 'NULL') + 13465*CalorietoJConversion
print('The enthalpy of {} is {}kJ'.format('reaction 2', round(
        EnthalpyforReaction2 * 1e-3,2)))

ExpectedEnthalpyReaction1 = 5477.04 * R - 16.64 * R * T
ExpectedEnthalpyReaction2 = 7290.75 * R - 6.09 * R * T

ErrorInEnthalpy(EnthalpyforReaction1, 'reaction 1', ExpectedEnthalpyReaction1)
```

```
ErrorInEnthalpy(EnthalpyforReaction2, 'reaction 2', ExpectedEnthalpyReaction2)
```