Oil Drop

Jason Pruitt

October 2019

1 Theory

As a drop of oil falls through air, it experiences a drag force before reaching its terminal velocity v_f. Turning on an electric field, the drop rises to terminal velocity v_u. We obtain the charge on the drop from eliminating the drag coefficient from the falling and rising net forces shown in Equations 1 and 2. The resulting equation for the charge on the drop is given by:

$$q_u = \frac{mg}{E}(\frac{v_u}{v_f} + 1) \tag{1}$$

2 Methods

Drops of oil were dispensed from a dropper into a chamber where they were under the influence of the electric field between two parallel plate capacitors. The field value was obtained with

$$E = \frac{V}{d}. \tag{2}$$

Reversing the voltage with a switch changed the direction the drops traveled, the drops' movement was timed to calculate velocity as they fell and rose. By rearranging Stoke's Law to solve for r,

$$r = \sqrt{\frac{9\eta v_f}{2g\rho}}, \tag{3}$$

where η is the viscosity of the medium and ρ is the viscosity of the oil, we calculated the drops' radii. Using the found radii values, we calculated the drops' masses with

$$m = \frac{4}{3}\pi r^3 \rho_{oil}. \tag{4}$$

Finally, each value was inserted into Equation 1 to obtain the charge accumulated by our drops.

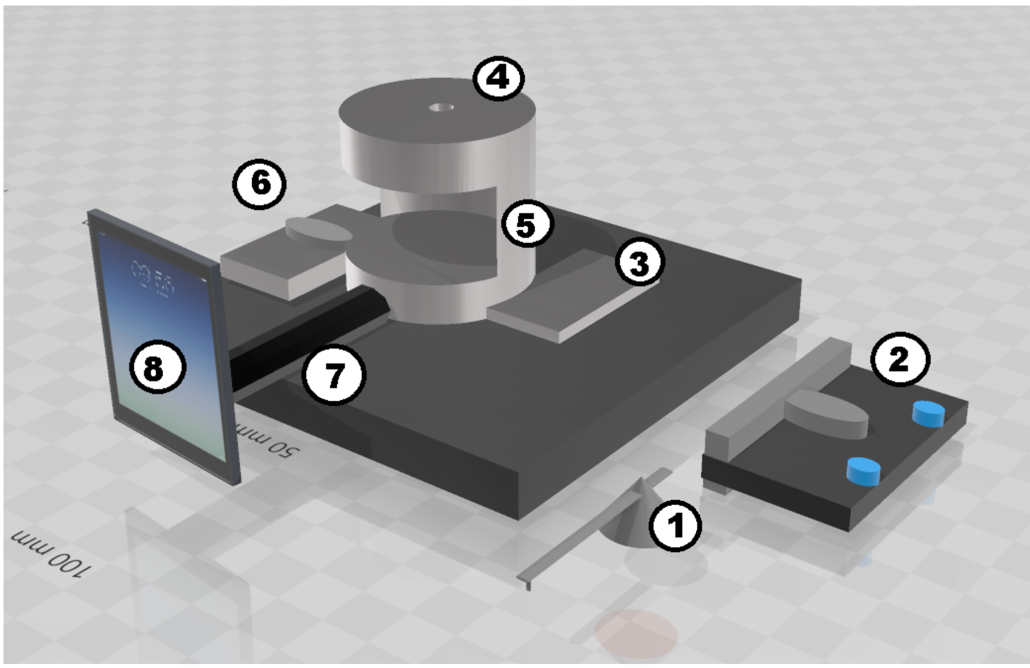

Figure 1: The setup and apparatus consisted of the oil dropper (1), the switch to change the direction of the electric field (2) between two parallel plate capacitors(5), a light source to illuminate the drops (3), a small hole at the top of the chamber (4), the switch to allow the drops to accumulate charge (6), and an iPad (8) to capture the footage from the viewing tube (7).

3 Data and Analysis

The uncertainty in velocity was calculated with

$$\delta V_{up} = v_{up} * \frac{time_{down}}{time_{up}}. \tag{5}$$

The uncertainty in η was calculated with

$$\delta\eta = \frac{\partial\eta}{\partial T} * \delta T. \tag{6}$$

The uncertainty in r was calculated with

$$\delta r = \sqrt{(\frac{\partial r}{\partial \eta})^2(\delta\eta)^2 + (\frac{\partial r}{\partial v_{down}})^2(\delta v_{down})^2}. \tag{7}$$

The uncertainty in m was calculated with

$$\delta m = |3m(\frac{\delta r}{r})|. \tag{8}$$

The uncertainty in γ was calculated with

$$\delta\gamma = |\frac{\partial\gamma}{\partial r}\delta r|. \tag{9}$$

Each of the aforementioned uncertainties were instrumental in calculating the uncertainty in q, which was calculated with

$$\delta q = \sqrt{\frac{\partial q}{\partial \gamma} + \frac{\partial q}{\partial m} + \frac{\partial q}{\partial v_{up}} + \frac{\partial q}{\partial v_{down}}} \tag{10}$$

Given the accepted value of e from NIST [1], and seen from the data in Table 10 and Figure 2, the q/e for each drop is within 2 σ of an integer value. Furthermore, as the value of charge increases, so does the uncertainty, which is in accordance to what we expect.

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
2.67 ± 0.05	9.75 ± 0.05	302 ± 2	2.196 ± 0.002
2.51 ± 0.05	7.22 ± 0.05		
3.16 ± 0.05	7.49 ± 0.05		
2.81 ± 0.05	9.11 ± 0.05		
2.69 ± 0.05	8.67 ± 0.05		
2.77 ± 0.05	9.73 ± 0.05		
2.87 ± 0.05	8.52 ± 0.05		
2.6 ± 0.05	7.09 ± 0.05		
2.64 ± 0.05	8.64 ± 0.05		

Table 1: Drop 1: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
3.11 ± 0.05	20.55 ± 0.05	300 ± 2	2.064 ± 0.002
2.85 ± 0.05	26.38 ± 0.05		
2.94 ± 0.05	24.6 ± 0.05		
2.59 ± 0.05	33.33 ± 0.05		
2.67 ± 0.05	22.47 ± 0.05		
1.29 ± 0.05	21.27 ± 0.05		
2.92 ± 0.05	20.34 ± 0.05		
2.93 ± 0.05	21.89 ± 0.05		

Table 2: Drop 2: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
0.73 ± 0.05	47.32 ± 0.05	300 ± 2	2.020 ± 0.002
0.95 ± 0.05	43.41 ± 0.05		

Table 3: Drop 3: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
1.52 ± 0.05	8.89 ± 0.05	299 ± 2	2.012 ± 0.002
1.37 ± 0.05	11.9 ± 0.05		
0.98 ± 0.05	11.86 ± 0.05		
0.94 ± 0.05	9.52 ± 0.05		
1.03 ± 0.05	11.75 ± 0.05		
0.87 ± 0.05	11.53 ± 0.05		
1.1 ± 0.05	10.71 ± 0.05		
1.13 ± 0.05	11.68 ± 0.05		

Table 4: Drop 4: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
0.9 ± 0.05	20.33 ± 0.05	299 ± 2	1.990 ± 0.002
1.0 ± 0.05	19.69 ± 0.05		
1.04 ± 0.05	18.72 ± 0.05		
0.92 ± 0.05	26.2 ± 0.05		
0.85 ± 0.05	26.08 ± 0.05		
0.96 ± 0.05	22.92 ± 0.05		
0.86 ± 0.05	20.87 ± 0.05		
0.94 ± 0.05	15.89 ± 0.05		

Table 5: Drop 5: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
3.75 ± 0.05	4.46 ± 0.05	299 ± 2	1.985 ± 0.002
4.06 ± 0.05	5.18 ± 0.05		
3.66 ± 0.05	5.66 ± 0.05		
3.27 ± 0.05	4.48 ± 0.05		
3.39 ± 0.05	5.43 ± 0.05		
3.52 ± 0.05	5.09 ± 0.05		
3.7 ± 0.05	4.64 ± 0.05		
3.12 ± 0.05	5.31 ± 0.05		

Table 6: Drop 6: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
2.55 ± 0.05	6.43 ± 0.05	299 ± 2	1.980 ± 0.002
2.9 ± 0.05	5.58 ± 0.05		
2.84 ± 0.05	6.41 ± 0.05		
2.58 ± 0.05	5.88 ± 0.05		

Table 7: Drop 7: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
1.15 ± 0.05	7.2 ± 0.05	299 ± 2	1.977 ± 0.002
2.01 ± 0.05	7.32 ± 0.05		
1.99 ± 0.05	8.86 ± 0.05		
1.52 ± 0.05	9.81 ± 0.05		
1.73 ± 0.05	6.61 ± 0.05		
1.67 ± 0.05	9.85 ± 0.05		
1.75 ± 0.05	6.87 ± 0.05		
2.01 ± 0.05	8.1 ± 0.05		

Table 8: Drop 8: Raw Data

t_{up} (s)	t_{down} (s)	$Voltage$ (V)	$Thermistor Resistance \times 10^6$ (Ω)
1.61 ± 0.05	13.04 ± 0.05	299 ± 2	1.975 ± 0.002
0.46 ± 0.05	18.7 ± 0.05		
0.65 ± 0.05	17.28 ± 0.05		
0.82 ± 0.05	16.5 ± 0.05		
0.5 ± 0.05	17.46 ± 0.05		
0.56 ± 0.05	17.97 ± 0.05		
0.54 ± 0.05	16.89 ± 0.05		
0.46 ± 0.05	13.9 ± 0.05		

Table 9: Drop 9: Raw Data

$q_u \times 10^{-19}[C]$	$\frac{q}{e} ratio$
(5.15 ± 0.07)	(3.21 ± 0.05)
(3.33 ± 0.07)	(2.08 ± 0.04)
(7.4 ± 0.4)	(4.6 ± 0.3)
(10.0 ± 0.4)	(6.2 ± 0.3)
(8.9 ± 0.4)	(5.6 ± 0.3)
(6.42 ± 0.07)	(4.01 ± 0.04)
(6.56 ± 0.09)	(4.10 ± 0.06)
(8.1 ± 0.2)	(5.05 ± 0.13)
(15.0 ± 1.3)	(9.4 ± 0.8)

Table 10: All Drops: Derived Data

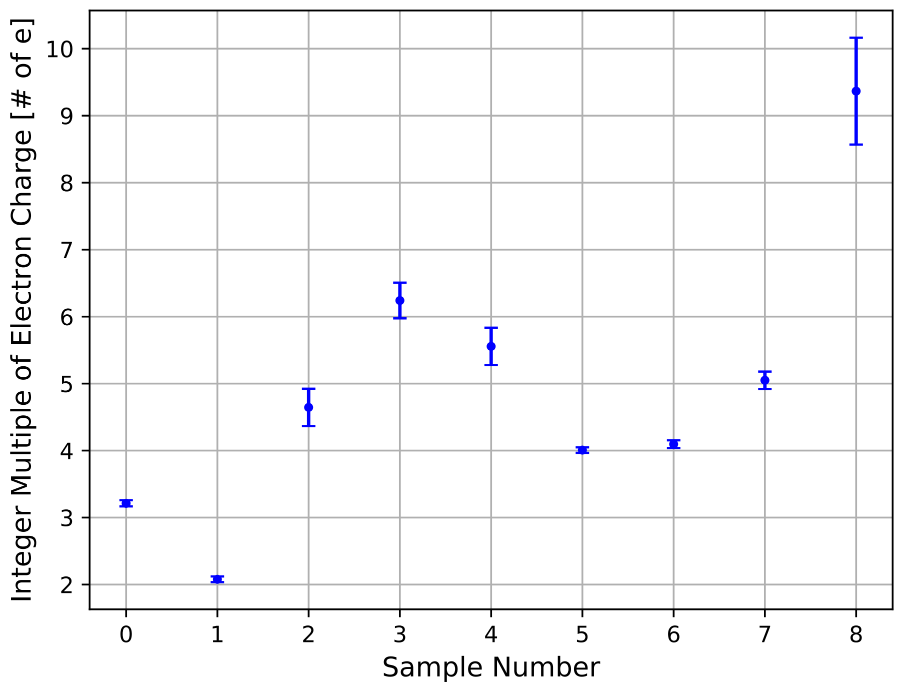

Figure 2: $\frac{q}{e} vs Sample Number$

References

[1] elementary charge. https://physics.nist.gov/cgi-bin/cuu/Value?e.

```
import numpy as np
import os
import matplotlib.pyplot as plt
from scipy import interpolate
import qLabMods
from pint import UnitRegistry

ureg = UnitRegistry()

def drop1():
    d = 2 # boxes
    d = d*0.1e-3 # m
    dist = 7.6e-3 # m
    downT = np.array
        ([9.75,7.22,7.49,9.11,8.67,9.73,8.52,7.09,8.64])
    upT = np.array
        ([2.67,2.51,3.16,2.81,2.69,2.77,2.87,2.60,2.64])
    dTime = 0.05 # sec
    V = 302 # V
    dV = 2
    tR = 2.196e6 # Ohms
    dtR = 0.002e6

    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop2():
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array
        ([20.55,26.38,24.6,33.33,22.47,21.27,20.34,21.89])
    upT = np.array([3.11,2.85,2.94,2.59,2.67,1.29,2.92,2.93])
    dTime = 0.05
    V = 300
    dV = 2
    tR = 2.064e6
    dtR = 0.002e6

    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop3():
```

```
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array([47.32,43.41])
    upT = np.array([0.73,0.95])
    dTime = 0.05
    V = 300
    dV = 2
    tR = 2.0206e6
    dtR = 0.002e6

    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop4():
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array
        ([8.89,11.9,11.86,9.52,11.75,11.53,10.71,11.68])
    upT = np.array([1.52,1.37,0.98,0.94,1.03,0.87,1.1,1.13])
    dTime = 0.05
    V = 299
    dV = 2
    tR = 2.012e6
    dtR = 0.002e6

    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop5():
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array
        ([20.33,19.69,18.72,26.2,26.08,22.92,20.87,15.89])
    upT = np.array([0.9,1.0,1.04,0.92,0.85,0.96,0.86,0.94])
    dTime = 0.05
    V = 299
    dV = 2
    tR = 1.99e6
    dtR = 0.002e6

    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop6():
    d = 2
```

```
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array([4.46,5.18,5.66,4.48,5.43,5.09,4.64,5.31])
    upT = np.array([3.75,4.06,3.66,3.27,3.39,3.52,3.7,3.12])
    dTime = 0.05
    V = 299
    dV = 2
    tR = 1.985e6
    dtR = 0.002e6

    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop7():
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array([6.43,5.58,6.41,5.88])
    upT = np.array([2.55,2.9,2.84,2.58])
    dTime = 0.05
    V = 299
    dV = 2
    tR = 1.98e6
    dtR = 0.002e6
    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop8():
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array([7.2,7.32,8.86,9.81,6.61,9.85,6.87,8.1])
    upT = np.array([1.15,2.01,1.99,1.52,1.73,1.67,1.75,2.01])
    dTime = 0.05
    V = 299
    dV = 2
    tR = 1.977e6
    dtR = 0.002e6
    return dist,d,downT,upT,dTime,V,dV,tR,dtR

def drop9():
    d = 2
    d = d*0.1e-3
    dist = 7.6e-3
    downT = np.array
        ([13.04,18.7,17.28,16.5,17.46,17.97,16.89,13.9])
    upT = np.array([1.61,0.46,0.65,0.82,0.5,0.56,0.54,0.46])
```

```
    dTime = 0.05
    V = 299
    dV = 2
    tR = 1.975e6
    dtR = 0.002e6
    return dist,d,downT,upT,dTime,V,dV,tR,dtR


def calculate_velocities(d,downT,upT,dTime):
    vDown = d/downT
    vDownBar = np.mean(vDown)
    dvDown = vDown*(dTime/downT)
    vUp = d/upT
    vUpBar = np.mean(vUp)
    dvUp = vUp*(dTime/upT)
    # print(vFall,vRise)
    # print(vFallBar,vRiseBar)

    return(vDown,vDownBar,dvDown,vUp,vUpBar,dvUp)

def interp_T(tR,dtR):

    C = np.array([17,18,19,20,21,22,23,24,25,26,27,28,29])
    R = np.array
        ([2.526,2.446,2.371,2.3,2.233,2.169,2.11,2.025,2.00,1.950,1.902,

                1.857,1.815])*1e6

    f = interpolate.interp1d(R,C)
    temp = f(tR)

    return temp

def calculate_q(dist,vDown,vDownBar,dvDown,vUp,vUpBar,dvUp,volts
    ,dTime,T):
    g = 9.8
    rho = 885 # kg/m^3
    # T = 22.5 # Celsius, change later
    dTemp = .3
    b = 0.82e-7

    E = volts/dist
    e = 1.6021766208e-19

    eta = (1.800 + ((T-15)/209))*10**-5
```

```
    dEtadT = (10**-5)/209
    etaUnc = dEtadT*dTemp

    r = np.sqrt((9*eta*vDownBar)/(2*g*rho))
    drdEta = (9*vDownBar)*(etaUnc**2)/(8*g*rho*eta)
    drdvDown = (9*eta)*(dvDown**2)/(8*g*rho*vDownBar)
    rUnc = ((drdEta**2)*(etaUnc**2) + (drdvDown**2)*(dvDown**2))
        **.5

    m = (4/3)*(np.pi*r**3*rho)
    mUnc = np.abs(3*m*(rUnc/r))

    gamma = (1/(1+(b/r)))**(-3/2)
    dGammadr = (3/2)*(-b/(r**2))*(1 + (b/r))**.5
    gammaUnc = np.abs(dGammadr*rUnc)

    qU = gamma*(m*9.8/E)*(vUp/vDown + 1)
    qU = np.mean(qU)
    dqdGamma = (((m*g)/E)*(vUpBar/vDownBar + 1)*gammaUnc)**2
    dqdm = (((gamma*g)/E)*(vUpBar/vDownBar + 1)*mUnc)**2
    dqdvUp = (((gamma*m*g)/(E*vDownBar))*dvUp)**2
    dqdvDown = (((gamma*m*g*vUp)/(E*(vDown**2)))*dvDown)**2
    dq = np.sqrt(dqdGamma + dqdm + dqdvUp + dqdvDown)

    ratio = qU/e
    ratioUnc = np.mean(dq)*(1/e)

    # print('\nq',qU)
    # print('dq/e:',ratioUnc)
    # print('q/e',ratio)

    return qU,ratio,dq,ratioUnc,r,rUnc,eta,etaUnc,m,mUnc,gamma,
        gammaUnc

def plots(ratio,uncs):

    samples = np.arange(len(ratio))
    plt.figure()
    plt.errorbar(samples,ratio,yerr=uncs,fmt = 'b.',capsize = 3)
    plt.xlabel('Sample Number',fontsize = 12)
    plt.ylabel('Integer Multiple of Electron Charge [# of e]',
        fontsize = 12)
    plt.grid()
    plt.savefig('oilDrop.pdf')
    # plt.show()
```

```
def main():
    dropList = [drop1,drop2,drop3,drop4,drop5,drop6,drop7,drop8,
        drop9]
    drops = len(dropList)

    etaList = []
    etaUncs = []
    mList = []
    mUncs = []
    gammaList = []
    gammaUncs = []
    rList = []
    rUncs = []
    qList = []
    dqList = []
    ratioList = []
    ratioUncs = []
    for ind in np.arange(drops):
        # print('\nDrop {}'.format(ind+1))
        (dist,d,downT,upT,dTime,volts,dV,tR,dtR) = dropList[ind
            ]()

        temp = interp_T(tR,dtR)

        (vDown,vDownBar,dvDown,vUp,vUpBar,dvUp) =
            calculate_velocities(d,downT,
                                                                        upT
                                                                            ,
                                                                            dTime
                                                                            )

        (qU,ratio,dq,ratioUnc,r,rUnc,
        eta,etaUnc,m,mUnc,gamma,gammaUnc) = calculate_q(dist,
            vDown,vDownBar,
                                                        dvDown,
                                                            vUp,
                                                            vUpBar
                                                            ,dvUp
                                                            ,
                                                        volts,
                                                            dTime
                                                            ,temp
                                                            )
```

```
etaList.append(eta)
etaUncs.append(etaUnc)
mList.append(m)
mUncs.append(np.mean(mUnc))
gammaList.append(gamma)
gammaUncs.append(gammaUnc)
rList.append(r)
rUncs.append(np.mean(rUnc))
qList.append(qU)
dqList.append(np.mean(dq))
ratioList.append(ratio)
ratioUncs.append(ratioUnc)

dTime = dTime*np.ones(len(upT))
varray = [volts]
dVarray = [dV]
tRarray = [tR]
dtRarray = [dtR]
# qArray = [qU]
# dqArray = [np.mean(dq)]
# ratioArray = [ratio]
# ratioUncArray = [ratioUnc]
# etaArray = [eta]
# etaUncArray = [etaUnc]
# mArray = [m]
# mUncArray = [mUnc]
# rArray = [r]
# rUncArray = [np.mean(rUnc)]
# gammaArray = [gamma]
# gammaUncArray = [np.mean(gammaUnc)]

for item in upT:
    varray.append([])
    dVarray.append([])
    tRarray.append([])
    dtRarray.append([])
    # qArray.append([])
    # dqArray.append([])
    # ratioArray.append([])
    # ratioUncArray.append([])
    # etaArray.append([])
    # etaUncArray.append([])
    # mArray.append([])
    # mUncArray.append([])
```

```
        # rArray.append([])
        # rUncArray.append([])
        # gammaArray.append([])
        # gammaUncArray.append([])

    # print(ratioArray)
    # print(ratioUncArray)
    # print('dq:',dqArray)
    # print('dvup:',dvUpArray)
    # print('dvdown:',dvDownArray)
    # print('lencheck',len(vUp),len(dvUpArray))
    # print('eta',etaArray)
    # print('etaunc',etaUncArray)
    # print('r',rArray)
    # print('runc',rUncArray)
    # print('gamma',gammaArray)
    # print(gammaUncArray)
    # print('q',qArray)
    # print('dq',dq)
    # print('ratio',ratioArray)
    # print(ratioUncArray)
    # print('lencheck',len(qArray)==len(dqArray))
    # print('lencheck',len(ratioArray)==len(ratioUncArray))
    table1 = qLabMods.LaTeXTable(['$t_{up}$','$t_{down}$','
        $Voltage$','Thermistor Resistance'],
                                    [(upT,dTime,ureg.seconds),
                                    (downT,dTime,ureg.seconds),
                                    (varray,dVarray,ureg.volts),
                                    (tRarray,dtRarray,ureg.ohms)
                                        ],
                                    'Drop {}: Raw Data'.format(
                                        ind+1),
                                    'DataTable',
                                    '!htp')

    # print(table1.allTogetherRaw())


# table2 = qLabMods.LaTeXTable(['$eta$','$\gamma$','$m$','
    $radius$','$q_{u}$','$q/e ratio$'],
#                                   [(etaList,etaUnc,ureg.seconds
    ),
#                                   (gammaList,gammaUncs,ureg.
    seconds),
#                                   (mList,mUncs,ureg.kilogram),
```

```
    #                                       (rList , rUncs , ureg . meter ) ,
    #                                       ( qList , dqList , ureg . meter/ureg
        . seconds ) ,
    #                                       ( ratioList , ratioUncs , ureg .
        seconds ) ] ,
    #                                       'All Drops: Derived Data ',
    #                                       'DataTable ',
    #                                       '!htp ')
    plots ( ratioList , ratioUncs )
    # print ( table2 . allTogetherSci ( ) )

main ( )
```