Charge To Mass Ratio: Week Two

Jason Pruitt

October 2019

1 Introduction

In the current day, the electron's mass [1] and charge [2] are well known. From introductory physics we learn that an electron traveling through a magnetic field is deflected by a force perpendicular to its motion. [3] Given this relationship, we sought to reproduce the results of J.J. Thomson's investigation of the charge-to-mass ratio [4].

By accelerating a beam of electrons through the magnetic field produced by a Helmholtz coil, we measured the deflection's radius of curvature in relation to the intensity of the field, and extrapolated the value of the electron's charge-to-mass ratio.

2 Data and Analysis

1. The raw x was plotted against the average between the positive and negative y values (to eliminate magnetic field offset) and a model, Equation 1, was fitted to the data.

$$y = y_0 - \sqrt{r^2 - (x - x_0)^2} \tag{1}$$

From the model, we extracted the radius of curvature r. The B field generated was calculated by Equation 2.

$$B = \frac{16\mu_0 NI}{\sqrt{125}D} \tag{2}$$

$x \times 10^1$ (cm)	$yAvg \times 10^{-2}$(cm)	$I \times 10^{-3}(A)$	$V_a(kV)$	$r \times 10^{-1}$(cm)	$\frac{e}{m}(\frac{C}{kg})$
(0.90 ± 0.10)	(2.12 ± 0.16)	(688.0 ± 0.1)	$(2.98 \pm .01)$	(2.37 ± 0.14)	(1.7 ± 0.2)
(0.80 ± 0.10)	(1.70 ± 0.16)				
(0.70 ± 0.10)	(1.30 ± 0.16)				
(0.60 ± 0.10)	(1.00 ± 0.16)				
(0.50 ± 0.10)	(0.70 ± 0.16)				
(0.40 ± 0.10)	(0.46 ± 0.16)				
(0.30 ± 0.10)	(0.31 ± 0.16)				
(0.20 ± 0.10)	(0.13 ± 0.16)				
(0.90 ± 0.10)	$(1.40 \pm .11)$	(465.0 ± 0.1)	$(2.98 \pm .01)$	(3.8 ± 0.5)	(1.5 ± 0.3)
(0.80 ± 0.10)	$(1.10 \pm .11)$				
(0.70 ± 0.10)	$(0.90 \pm .11)$				
(0.60 ± 0.10)	$(0.70 \pm .11)$				
(0.50 ± 0.10)	$(0.49 \pm .11)$				
(0.40 ± 0.10)	$(0.31 \pm .11)$				
(0.30 ± 0.10)	$(0.22 \pm .11)$				
(0.20 ± 0.10)	$(0.11 \pm .11)$				
(0.90 ± 0.10)	$(1.60 \pm .12)$	(545.0 ± 0.1)	$(2.98 \pm .01)$	(3.5 ± 0.4)	(1.3 ± 0.3)
(0.80 ± 0.10)	$(1.30 \pm .12)$				
(0.70 ± 0.10)	$(1.00 \pm .12)$				
(0.60 ± 0.10)	$(1.00 \pm .12)$				
(0.50 ± 0.10)	$(0.70 \pm .12)$				
(0.40 ± 0.10)	$(0.46 \pm .12)$				
(0.30 ± 0.10)	$(0.31 \pm .12)$				
(0.20 ± 0.10)	$(0.13 \pm .12)$				
(0.90 ± 0.10)	$(0.90 \pm .10)$	(307.0 ± 0.1)	$(2.98 \pm .01)$	(5.2 ± 0.9)	(1.8 ± 0.6)
(0.80 ± 0.10)	$(0.90 \pm .10)$				
(0.70 ± 0.10)	$(0.72 \pm .10)$				
(0.60 ± 0.10)	$(0.44 \pm .10)$				
(0.50 ± 0.10)	$(0.31 \pm .10)$				
(0.40 ± 0.10)	$(0.19 \pm .10)$				
(0.30 ± 0.10)	$(0.12 \pm .10)$				
(0.20 ± 0.10)	$(0.09 \pm .10)$				
(0.90 ± 0.10)	$(1.32 \pm .10)$	(455.0 ± 0.1)	$(2.98 \pm .01)$	(4.0 ± 0.6)	(1.5 ± 0.4)
(0.80 ± 0.10)	$(1.09 \pm .10)$				
(0.70 ± 0.10)	$(0.85 \pm .10)$				
(0.60 ± 0.10)	$(0.63 \pm .10)$				
(0.50 ± 0.10)	$(0.49 \pm .10)$				
(0.40 ± 0.10)	$(0.30 \pm .10)$				
(0.30 ± 0.10)	$(0.14 \pm .10)$				
(0.20 ± 0.10)	$(0.11 \pm .10)$				

Table 1: Raw position coordinates, current, accelerating potential, derived radius of curvature and $\frac{e}{m}$

Where μ_0 was the permeability of free space, I was the current through the Helmholtz coils, N was the number of turns in the coils, and D was the diameter. The accelerating potential,earlier found r, and newly found B were inserted into Equation 3 to extrapolate the charge-to-mass ratio.

$$\frac{e}{m} = \frac{2V_a}{B^2r^2} \tag{3}$$

2. The average $\frac{e}{m}$ was found to be $(1.5 \pm .4) \times 10^{11} \frac{C}{kg}$. With the accepted value of $1.76 \times 10^{11} \frac{C}{kg}$, we concluded that our data was reasonable since the uncertainty overlaps with the accepted value.

   The two methods of obtaining $\delta\frac{e}{m}$ were propagating uncertainty and the standard deviation of the mean, *SDOM*. For propagating uncertainty, $\delta\frac{e}{m}$ was found from Equation 4.

$$\delta\frac{e}{m} = \frac{e}{m}\sqrt{(\frac{\delta V_a}{V_a})^2 + 4(\frac{\delta B}{B})^2 + 4(\frac{\delta r}{r})^2} \tag{4}$$

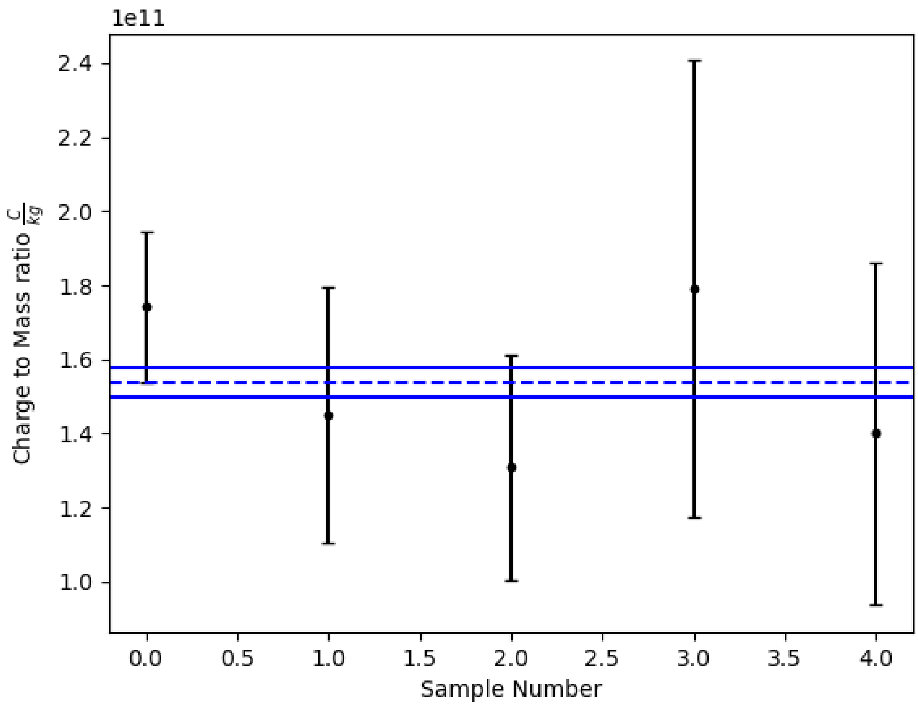
   We found that the propagation of uncertainty yielded a $\delta\frac{e}{m}$ that was lower than the overall SDOM for our first three measurements, which can be seen when comparing Table 1 and Table 2.

3. Table 2 displays the fact that more measurements lead to a lower SDOM.

Number of Runs	$\frac{\sigma(\frac{e}{m})}{\sqrt{N}}$
1	0.0
2	7.28×10^{-9}
3	6.01×10^{-9}
4	5.01×10^{-9}
5	3.84×10^{-9}

Table 2: Number of Runs and the SDOM

Figure 1: Plot of $\frac{e}{m}$ ratio vs sample number and mean value $\pm$ SDOM



Listing 1: Code

```
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy import stats
from scipy.optimize import curve_fit
# import qLabMods
# from pint import UnitRegistry
# ureg = UnitRegistry()

def run1():
    I = .688
    dI = 0.1e-3
    V = 2.98e3
    dV = .01e3
    xBoth = np.array([9,8,7,6,5,4,3,2])
    dxPos = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yPos = np.array([2,1.6,1.2,.9,.6,.31,.22,.04])
    dyPos = np.array([.01,.01,.05,.05,.05,.05,.05,.01])
    dxNeg = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yNeg = np.array([-2.25,-1.8,-1.4,-1.1,-0.8,-0.6,-0.4,-.21])
    dyNeg = np.array([0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025])

    return(I,dI,V,dV,xBoth,dxPos,yPos,dyPos,dxNeg,yNeg,dyNeg)

def run2():
    I = .465
    dI = 0.1e-3
    V = 2.97e3
    dV = .01e3
    xBoth = np.array([9,8,7,6,5,4,3,2])
    dxPos = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yPos = np.array([1.3,1.0,.8,.6,.4,.22,.18,.02])
    dyPos = np.array([0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025])
    dxNeg  = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yNeg = np.array([-1.5,-1.2,-1.0,-0.8,-.58,-.4,-.25,-.2])
    dyNeg = np.array([0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025])
    return(I,dI,V,dV,xBoth,dxPos,yPos,dyPos,dxNeg,yNeg,dyNeg)

def run3():
    I =.545
    dI = 0.1e-3
    V = 2.97e3
    dV = .01e3
    xBoth = np.array([9,8,7,6,5,4,3,2])
```

```python
    dxPos = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yPos = np.array([1.5,1.2,.9,.7,.5,.3,.18,.02])
    dyPos = np.array([0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025])
    dxNeg = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yNeg = np.array([-1.7,-1.4,-1.1,-.9,-.63,-.45,-.38,-.2])
    dyNeg = np.array([0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03])

    return(I,dI,V,dV,xBoth,dxPos,yPos,dyPos,dxNeg,yNeg,dyNeg)

def run4():
    I = .307
    dI = 0.1e-3
    V = 2.97e3
    dV = .01e3
    xBoth = np.array([9,8,7,6,5,4,3,2])
    dxPos = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yPos = np.array([.8,.6,.5,.38,.22,.08,.03,.001])
    dyPos = np.array([0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03])
    dxNeg  = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yNeg = np.array([-1.0,-.83,-.7,-.5,-.4,-.3,-.2,-.18])
    dyNeg = np.array([0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03])

    return(I,dI,V,dV,xBoth,dxPos,yPos,dyPos,dxNeg,yNeg,dyNeg)

def run5():
    I = .455
    dI = .01e-3
    V = 2.97e3
    dV = .01e3
    xBoth = np.array([9,8,7,6,5,4,3,2])
    dxPos = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yPos = np.array([1.2,.99,.75,.56,.4,.21,.06,.02])
    dyPos = np.array([0.025, 0.025, 0.025, 0.025, 0.025, 0.025, 0.025,
                      0.025])
    dxNeg = np.array([.01,.01,.01,.01,.01,.01,.01,.01])
    yNeg = np.array([-1.45,-1.2,-.95,-.7,-.58,-.4,-.22,-.2])
    dyNeg = np.array([0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03])

    return(I,dI,V,dV,xBoth,dxPos,yPos,dyPos,dxNeg,yNeg,dyNeg)


def plot_Calc(I,dI,V,dV,xBoth,dxPos,dxNeg,yAvg,dyAvg,posB,negB,dB,runNum):
    print('\n\tRun number',runNum+1)

    # Curves to fit for both pos and neg
```

```python
def funcPos(x,r,x0,y0):
    return (y0 - np.sqrt(r**2 - (x-x0)**2))

def funcNeg(x,r,x0,y0):
    return (y0 + np.sqrt(r**2 - (x-x0)**2))

# Give it an interpolated interval
xtheory = np.linspace(2,9)

# Initial guess for curve_fit func
p0 = [15/100,0,15/100]
popt,pcov = curve_fit(funcPos,xBoth,yAvg,p0,sigma=dxPos)

# Uncertainty in r comes from diagonal of cov matrix
perr = np.sqrt(np.diag(pcov))
# perrNeg = np.sqrt(np.diag(pcovNeg))

r = popt[0]
dr = perr[0]

cM = 2*V / ((posB**2)*(r**2))
dCMdV = 2/((posB**2)*(r**2))
dCMdB = -4*V / ((r**2)*(posB**3))
dCMdr = -4*V / ((posB**2)*(r**3))
dCM = np.sqrt(
            (dCMdV**2)*dV**2
          + (dCMdB**2)*dB**2
          + (dCMdr**2)*dr**2)


# plt.figure()
# plt.errorbar(xBoth,yAvg,yerr=dyPos,fmt='k.')
# plt.plot(xtheory,funcPos(xtheory,15/100,0,15/100),'b--')
# plt.title('Guess for Theory Curve')
# plt.xlabel('x[m]')
# plt.ylabel('y[m]')
# plt.show()
#
# plt.figure()
# plt.errorbar(xBoth,yAvg,xerr=dxPos,yerr=dyAvg,fmt='k.',capsize =4)
# plt.plot(xBoth,funcPos(xBoth,popt[0],popt[1],popt[2]),'k--')
# plt.xlabel('x[m]')
# plt.ylabel('y[m]')
# plt.savefig('chargeMassRun{}.png'.format(runNum+1))
# plt.show()
```

```
    return(cM,dCM,popt,r,dr)


def methods(vals,cmUnc,runs):

    meanCM = np.mean(vals)
    samples = np.arange(runs)
    sdom = np.std(vals)/len(vals)

    plt.figure()
    plt.plot(samples,vals,'k.')
    plt.errorbar(samples,vals,yerr=cmUnc,fmt = 'k.',capsize = 3)
    plt.axhline(meanCM,color = 'b',linestyle = '--')
    plt.axhline(meanCM+sdom,color = 'b')
    plt.axhline(meanCM-sdom,color = 'b')
    plt.ylabel('Charge to Mass ratio $\\frac{C}{kg}$')
    plt.xlabel('Sample Number')
    plt.savefig('cmratioVsSample.png')
    # plt.show()

def main():

    # Static Variables
    muNot = 1.25663706e-6 # N/A^2
    N = 131 # turns
    dN = .1
    D = 20.8e-2 # m
    dD = 0.001 # m

    # List of runs where elements are function handles
    runList = [run1,run2,run3,run4,run5]
    runs = len(runList)

    # Preallocate empty list to later compare all c/m ratios to each other
    cmRatioList = []
    cmUncList = []

    # Loop over number of runs
    plt.figure()
    for ind in np.arange(runs):

        # Call func number at index number to call the correct number
        # Vars are redefined every pass through the loop
        I,dI,V,dV,xBoth,dxPos,yPos,dyPos,dxNeg,yNeg,dyNeg = runList[ind]()
```

```
# Convert to m
xBoth = xBoth/100
dxPos = dxPos/100
yPos = yPos/100
dyPos = dyPos/100
dxNeg = dxNeg/100
yNeg = yNeg/100
dyNeg = dyNeg/100

yAvg = []
for pos,neg in zip(yPos,yNeg):
    yAvg.append((pos + np.abs(neg))/2)
dyAvg = np.std(yAvg) / np.sqrt(len(yPos)+len(yNeg))

# B field takes in data from run
posB = (16*muNot*N*I)/(np.sqrt(125)*D) # T
negB = (16*muNot*N*-I)/(np.sqrt(125)*D) # T
dB = posB*np.sqrt(((dN/N)**2) + ((dI/I)**2) + ((dD/D)**2))

cM,dCM,popt,r,dr = plot_Calc(I,dI,V,dV,xBoth,dxPos,dxNeg,yAvg,dyAvg,posB,
                             negB,dB,ind)

cmRatioList.append(cM)
cmUncList.append(dCM)

def funcPos(x,r,x0,y0):
     return (y0 - np.sqrt(r**2 - (x-x0)**2))


plt.errorbar(xBoth,yAvg,xerr=dxPos,yerr=dyAvg,fmt='k.',capsize =4)
plt.plot(xBoth,funcPos(xBoth,popt[0],popt[1],popt[2]),'k--')
plt.xlabel('x[m]')
plt.ylabel('y[m]')
# plt.savefig('chargeMassRun{}.png'.format(ind+1))


print('xdata',xBoth)
print('xUnc',dxPos)
print('yavg:',yAvg)
print('dyAvg:',dyAvg)
print('r:',r)
print('dr:',dr)
print('e/m',cM)
print('e/m_unc',dCM)
```

```
    plt.show()


    methods(cmRatioList, cmUncList, runs)

main()
```

References

[1] electron mass. https://physics.nist.gov/cgi-bin/cuu/Value?me—search$_for = abbr_in$!

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

[3] Lorentz force. https://www.britannica.com/science/Lorentz-force.

[4] J.j. thomson's experiment and the charge-to-mass ratio of the electron. https://www.nyu.edu/classes/tuckerman/adv.chem/lectures/lecture$_3/node1.html.Accessed : 2019 - 10 - 10$.