PHYS 229 - Ryan Kaufmann/Experiment 2/Fields Simulations - Lab

SIGNED by Ryan Kaufmann Mar 23, 2018 @04:58 PM PDT

Ryan Kaufmann Mar 07, 2018 @06:02 PM PST

Field Simulations 1 - Investigating 'Near Field'

Ryan Kaufmann Mar 22, 2018 @03:20 PM PDT

In order to minimize our uncertainties with this experiment, we decided to try a python script. The python script creates a potential field of a quadrupole with two positive 1 nC charges and two negative 1 nC charges. The python script places our quadrupole for us such that it has a side length of 0.5 meters. Thus we can eliminate any uncertainty with the charges and the perfection of the quadrupole. Note that this doesn't take out all uncertainties. The data must be collected still with the mouse resulting in human error from inconsistencies when keeping a constant radius or angle. The file we used is:

Ryan Kaufmann Mar 22, 2018 @04:09 PM PDT

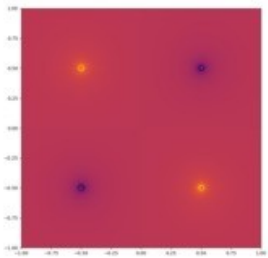
```
import numpy as np
import matplotlib.pyplot as plt

class Particle:
    coulombConstant = 8.988

    def __init__(self, xposition, yposition, charge):
        self.xpos = xposition
        self.ypos = yposition
        self.charge = charge

    def potential(self, xval, yval):
        distance = np.sqrt((self.xpos-xval)**2+(self.ypos-yval)**2)
        potential = self.coulombConstant*self.charge/distance
        excludeX = np.where(np.abs(np.abs(xval[0, :])-np.abs(self.xpos))<=0.01)
        excludeY = np.where(np.abs(np.abs(yval[:, 0])-np.abs(self.ypos))<=0.01)
        for x in excludeX[0]:
            for y in excludeY[0]:
                potential[x, y]=0
        return potential

def HalfCircle(x, r):
    return np.sqrt(r**2-x**2)

P1 = Particle(0.5, 0.5,  1)
P2 = Particle(-0.5, 0.5, -1)
P3 = Particle(0.5, -0.5, -1)
P4 = Particle(-0.5, -0.5, 1)

xCircle1 = np.linspace(-0.25, 0.25, 1000)
xCircle2 = np.linspace(-0.5,  0.5,  1000)
xCircle3 = np.linspace(-0.75, 0.75, 1000)
xCircle4 = np.linspace(-1.0,  1.0,  1000)
xvals = np.linspace(-1, 1, 1000)
yvals = np.linspace(-1, 1, 1000)
mXg, mYg = np.meshgrid(xvals, yvals)

potentialmap = P1.potential(mXg, mYg)
potentialmap += P2.potential(mXg, mYg)
potentialmap += P3.potential(mXg, mYg)
potentialmap += P4.potential(mXg, mYg)

plt.imshow(potentialmap, cmap='inferno', extent=[-1, 1, -1, 1])
plt.plot(xCircle1,  HalfCircle(xCircle1, 0.25), c='k')
plt.plot(xCircle1, -HalfCircle(xCircle1, 0.25), c='k')
plt.plot(xCircle2,  HalfCircle(xCircle2, 0.5),  c='k')
plt.plot(xCircle2, -HalfCircle(xCircle2, 0.5),  c='k')
plt.plot(xCircle3,  HalfCircle(xCircle3, 0.75), c='k')
plt.plot(xCircle3, -HalfCircle(xCircle3, 0.75), c='k')
plt.plot(xCircle4,  HalfCircle(xCircle4, 1.0),  c='k')
plt.plot(xCircle4, -HalfCircle(xCircle4, 1.0),  c='k')
plt.plot(xCircle4,  xCircle4, c='k')
plt.plot(xCircle4, -xCircle4, c='k')
plt.plot(xCircle4, np.zeros(xCircle4.shape), c='k')
plt.plot(np.zeros(xCircle4.shape), xCircle4, c='k')
plt.show()
```

FieldSimulation.py(2 KB)

Ryan Kaufmann Mar 22, 2018 @06:02 PM PDT

Which produces the following field simulation, with yellow representing positive potential and blue representing negative potential:

Ryan Kaufmann Mar 22, 2018 @05:43 PM PDT



FieldSimulation.png(55.1 KB)

Ryan Kaufmann Mar 22, 2018 @06:03 PM PDT

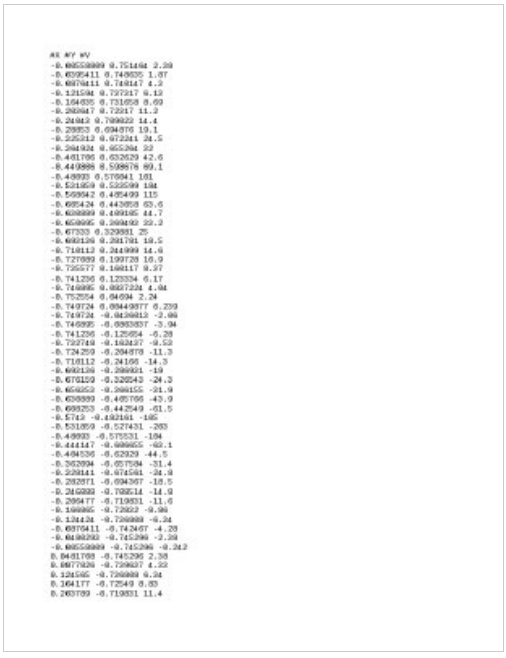
Our quadrupole was set up so that the top left and bottom right charges were positive (+1nC) and the top right and bottom left charges were negative (-1nC).

We then took data similar to the PhET simulator. Marking where our constant radius or angle is, we took various measurements of the potential. In the near field, which we defined to be less than about one quadrupole side length from the center. Since we set up our quadrupole to have a side length of 0.5 meters (which is the unit we're using to simulate this example), our near field stretches up to 1 meter away from the center. In order to simplify this an prevent having the measure strange quantities, we simply measured in a 1 meter by 1 meter square around the quadrupole, which is displayed above.

Our measurement kept either radius or angle constant. In total, we took eight sets of data, four of each type. Our constant angle measurements were at 0, pi/4, pi/2, and 3*pi/4 radians and went for both positive and negative radii. In these sets, we tried to measure every 0.03 meters to be consistent, but realized this wasn't feasible near the charges themselves, as the potential went to infinity. Our constant radius measurements were at radii of 0.25, 0.5, 0.75, and 1 meters and went from 0 to 2pi radians. In these sets, it was harder to try and measure at a constant angle change, but we decided to base our measurements instead in how fast the x or y coordinate changed (depending on which one was changing faster or rather which part of the circle we were measuring on). We attempted to measure, again, every 0.03 meters in the coordinate that was changing the fastest.

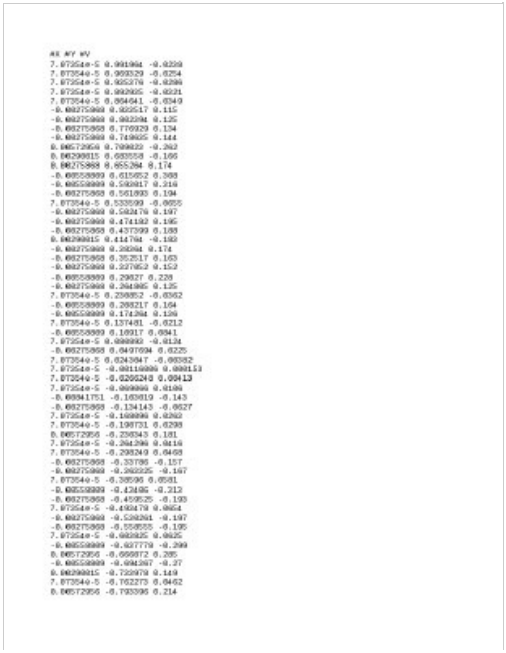
In the end, we got the following eight data sets:

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



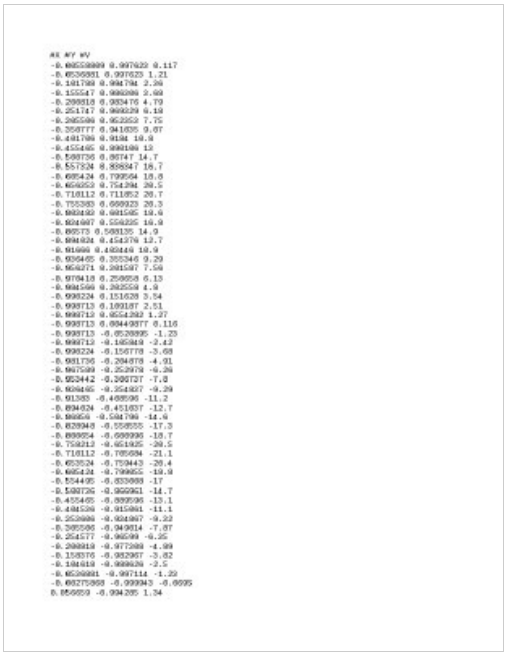
NearField075Radius.txt(2.6 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



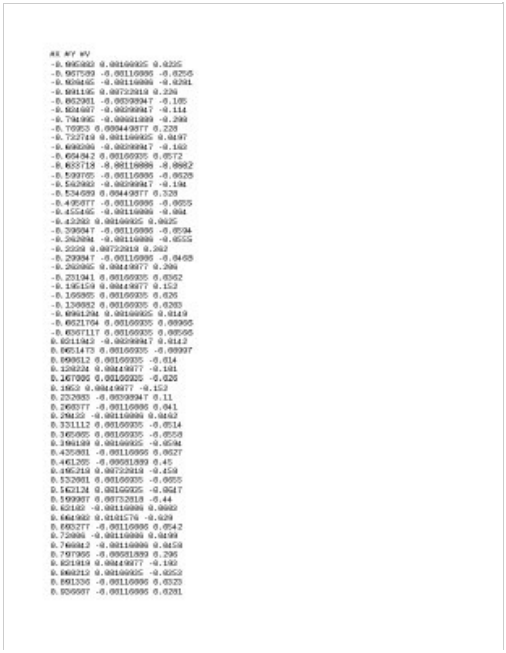
NearField90Vertical.txt(1.8 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



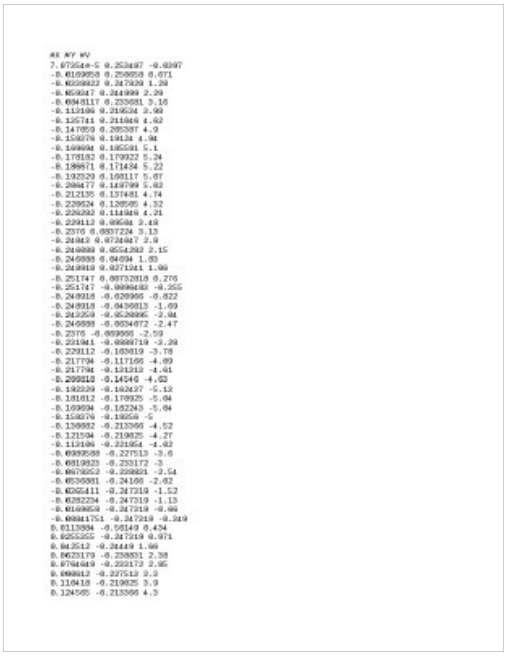
NearField100Radius.txt(2.8 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



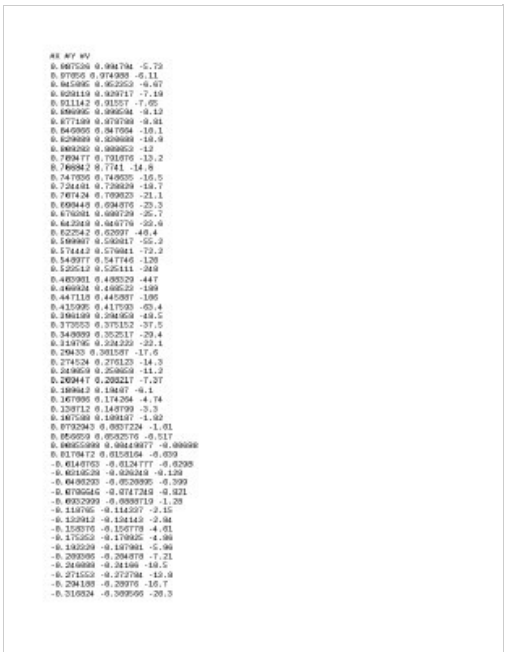
NearField00Horizontal.txt(1.7 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



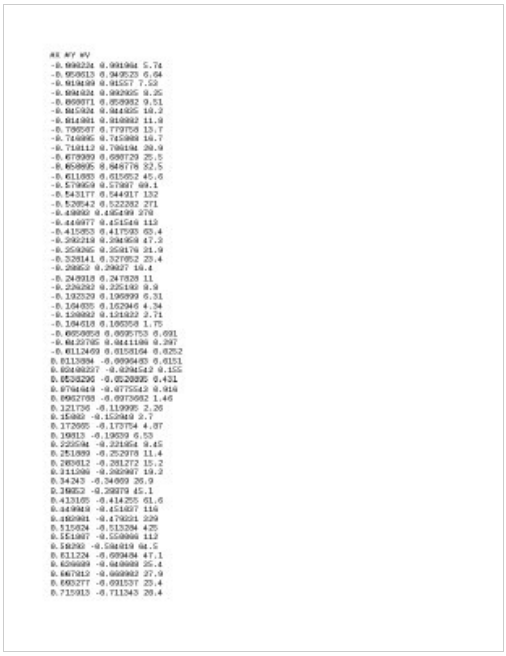
NearField025Radius.txt(2.4 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



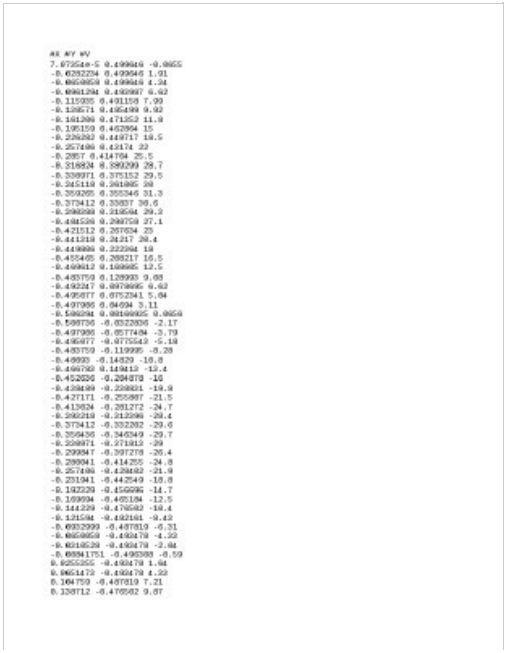
NearField45Negative.txt(2.1 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT



NearField45Positive.txt(1.6 KB)

Ryan Kaufmann Mar 22, 2018 @06:01 PM PDT

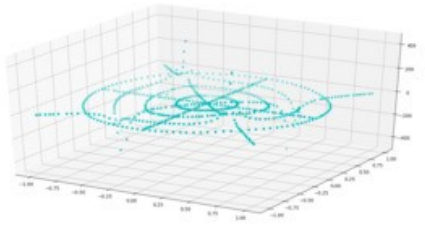


NearField050Radius.txt(2.5 KB)

Ryan Kaufmann Mar 22, 2018 @06:07 PM PDT

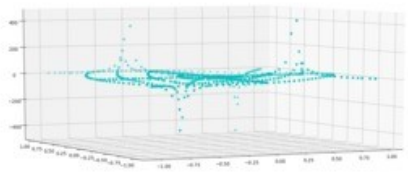
When then plotted all of our data in a 3-dimensional plot to get a look at it. We got the following result:

Ryan Kaufmann Mar 22, 2018 @06:07 PM PDT



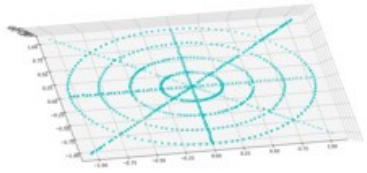
3DScatterPlotView1.png(241.4 KB)

Ryan Kaufmann Mar 22, 2018 @06:07 PM PDT



3DScatterPlotView2.png(132.4 KB)

Ryan Kaufmann Mar 22, 2018 @06:07 PM PDT



3DScatterPlotView3.png(222.9 KB)

Ryan Kaufmann Mar 23, 2018 @04:02 PM PDT

For the moment, we will not try to fit this data, as we can tell the fit is most likely a difficult equation. We will leave the fit for the section where we build a model.

Ryan Kaufmann Mar 07, 2018 @06:02 PM PST

Field Simulations 2 - Investigating 'Far Field' Distance Dependence

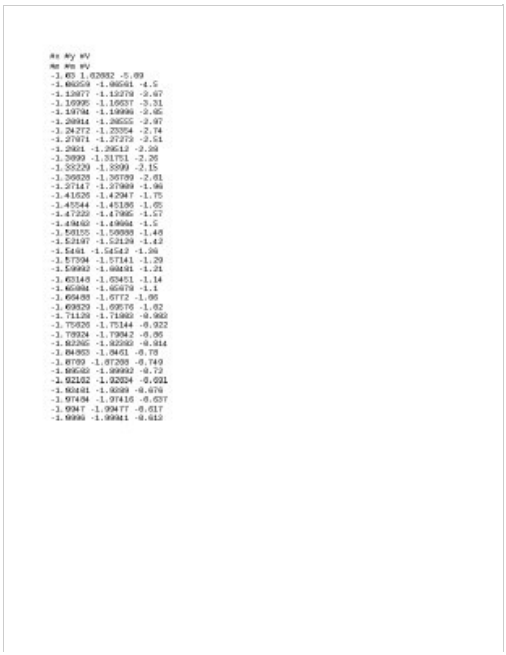
Ryan Kaufmann Mar 22, 2018 @07:41 PM PDT

After looking at the near field potentials, we can start also looking at the far field potentials. The far field potentials can be split into two relations, one with a distance dependence, and a second with an angular dependence. This section will focus on the distance dependency. In these next two sections, we set up our far field to be defined up to four times the side length away from the center. In other words, we measured in a 2 meter by 2 meter square. We limited this measurement more than the near field measurement. Here, we decided to take measurements along the pi/4 radian angle in the third quadrant (which has a negative charge).

In this section, when we take our uncertainties, we take them based on the divergence from the pi/4 angle constant. We set up our uncertainty to be a relative uncertainty in voltage then multiplied by a constant to make as close to 66.6% of our data to be within one uncertainty of the fit. This constant is required since we do not know the relationship between voltage and angle.

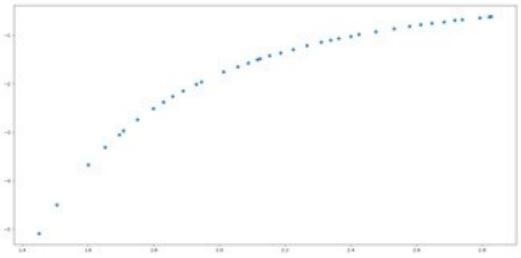
We repeat a similar process to our radial measurements for the near field. We measure along the pi/4 radian from the edge of the square (sqrt(8) away from the center) to the edge of the near field (sqrt(2) away from the center). We get the following data.

Ryan Kaufmann Mar 22, 2018 @07:02 PM PDT



FarFieldDistanceDependence.txt(960 Bytes)

Ryan Kaufmann Mar 23, 2018 @12:07 AM PDT



RadiusStraightPlot.png(20.7 KB)

Ryan Kaufmann Mar 22, 2018 @07:12 PM PDT

After looking at the data, we notice that it is most likely that the data has either a power or exponential dependence. We first tried a power law dependence. The following script was then used to attempt to curve fit the data to the following equation:

Ryan Kaufmann Mar 22, 2018 @07:11 PM PDT

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit as cf

def farRadiusFunc(radius, coeff, power, offset):
    return coeff*(radius)**(power)+offset

def chiSquared(x, y, sigma, params):
    return np.sum((y-farRadiusFunc(x, *params))**2/sigma**2)

filename = "C:/Users/ryank/Desktop/Work/Classes/Python/PHY229A/"
filename += "Field Simulations/Data/FarField2DistanceDependence.txt"
Data = np.loadtxt(filename)

DataSet = np.zeros(Data.shape)
DataSet[:, 0] = np.sqrt(Data[:, 0]**2+Data[:, 1]**2)
DataSet[:, 1] = Data[:, 2]
DataSet[:, 2] = (np.abs(np.arctan(Data[:, 1]/Data[:, 0]))-np.pi/4)/(np.pi/4)
DataSet[:, 2] *= Data[:, 2]*7.5
sigma = np.abs(np.average(DataSet[:, 2]))

##############################################################################

guess = [-1, -1, 0]
xtest = np.linspace(min(DataSet[:, 0]), max(DataSet[:, 0]), num=1000)

plt.errorbar(DataSet[:, 0], DataSet[:, 1], yerr=DataSet[:, 2], linestyle='',
             marker='.')
plt.plot(xtest, farRadiusFunc(xtest, *guess))
plt.show()

##############################################################################

Fitparams, Fitcov = cf(farRadiusFunc, DataSet[:, 0], DataSet[:, 1], p0=guess,
                       sigma=DataSet[:, 2])
ytest = farRadiusFunc(xtest, *Fitparams)

plt.errorbar(DataSet[:, 0], DataSet[:, 1], yerr=DataSet[:, 2], linestyle='',
             marker='.')
plt.plot(xtest, ytest)
plt.show()

chi = chiSquared(DataSet[:, 0], DataSet[:, 1], DataSet[:, 2], Fitparams)
dof = len(DataSet[:, 0])-len(Fitparams)
print ("\nGoodness of fit - Chi square measure:")
print ("Chi2 = {}, Chi2/dof = {}\n".format(chi, chi/dof))

Fitcov = Fitcov*dof/chi
paramSerr = np.sqrt(np.diag(Fitcov))

param_names = ['Coefficient', 'Power', 'Offset']

print ("Fit parameters:")
for i in range(len(Fitparams)):
    print ('{} = {:.3e} +/- {:.3e}'.format(param_names[i], Fitparams[i],
                                           paramSerr[i]))

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

CurveFitFarQuadrupoleRadius.py(2.8 KB)

Ryan Kaufmann Mar 22, 2018 @07:13 PM PDT

$$V = A * r^B + C$$

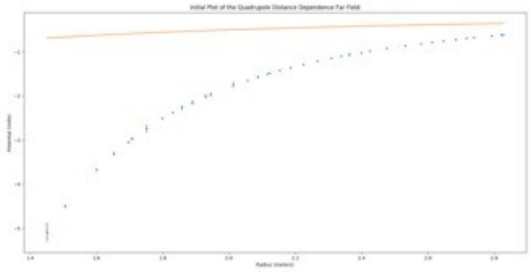
Ryan Kaufmann Mar 22, 2018 @07:13 PM PDT

Then plugging into the script, we get the following equation and graphs:

Ryan Kaufmann Mar 22, 2018 @07:16 PM PDT

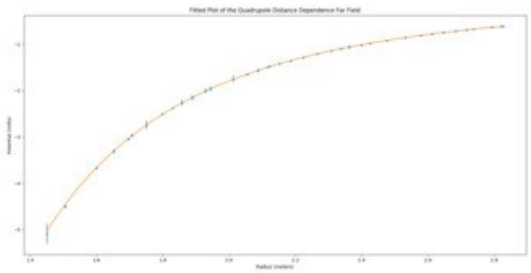
$$A = -16.7471 \pm 0.2043$$
$$B = -3.25985 \pm 0.02169$$
$$C = -0.047625 \pm 0.006018$$

Ryan Kaufmann Mar 22, 2018 @07:32 PM PDT



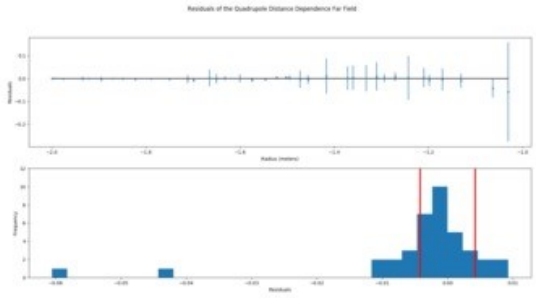
RadiusInitial.png(32.8 KB)

Ryan Kaufmann Mar 22, 2018 @07:32 PM PDT



RadiusFitted.png(44.7 KB)

Ryan Kaufmann Mar 22, 2018 @07:32 PM PDT



RadiusResids.png(36.6 KB)

Ryan Kaufmann Mar 22, 2018 @07:37 PM PDT

After fitting the function, we get a chi-squared of 0.505532. Our chi-squared is very good, that is well below 1, suggesting a good fit between the power series and the data. The residuals reinforce this, revolving well around 0 and having some random scatter. Thus, we can conclude that the relationship between radius and potential can be stated as the following (approximate) equation:

Ryan Kaufmann Mar 22, 2018 @07:38 PM PDT

$$V(r) = -16.7471 * r^{-3.25985} - 0.047625$$

Ryan Kaufmann Mar 22, 2018 @07:39 PM PDT

This result can be analyzed further by taking more data points and refining the definition of the uncertainties.

Ryan Kaufmann Mar 07, 2018 @06:02 PM PST

Field Simulations 3 - Investigating 'Far Field' Angular Dependence

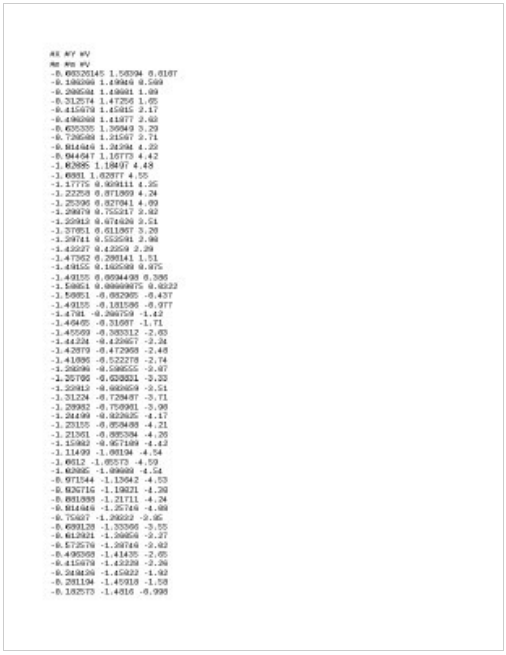
Ryan Kaufmann Mar 22, 2018 @11:47 PM PDT

In this section, we continue our look at the far field and attempt to fit a function to the angular dependency. Again we work in our 2 meter by 2 meter square. In this part, we are taking measurements at a constant radius of 1.5 meters. We only measure from 0 radians to pi radians, where 0 radians is the vertical. This half circle is a short cut due to symmetry. We realized that our configuration is repeated in the second half of the circle, thus any measurements would be repeated as well.

In this section, when we take our uncertainties, we take them based on the divergence from the 1.5 meter radius constant. We set up our uncertainty to be a relative uncertainty in voltage then multiplied by a constant to make as close to 66.6% of our data to be within one uncertainty of the fit.

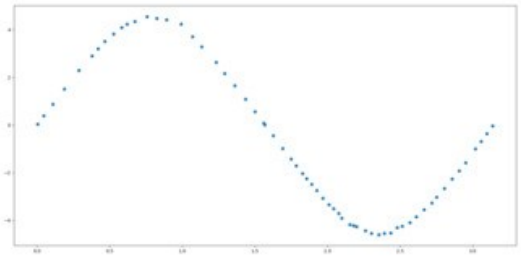
We repeat a similar process to our angular measurements for the near field. We measure along the 1.5 meters from the center from 0 radians to pi radians, getting the following data:

Ryan Kaufmann Mar 22, 2018 @11:47 PM PDT



FarFieldAngleDependence.txt(1.5 KB)

Ryan Kaufmann Mar 23, 2018 @12:08 AM PDT



ThetaStraightPlot.png(24 KB)

Ryan Kaufmann Mar 22, 2018 @11:55 PM PDT

The data is clearly a sine wave. Thus, we use the following curve fit to fit the function to the following sine wave:

Ryan Kaufmann Mar 22, 2018 @11:58 PM PDT

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit as cf

def farThetaFunc(theta, coeff, phase, shift, offset):
    return coeff*np.sin(phase*(theta+shift))+offset

def chiSquared(x, y, sigma, params):
    return np.sum((y-farThetaFunc(x, *params))**2/sigma**2)

filename = "C:/Users/ryank/Desktop/Work/Classes/Python/PHYS229/"
filename += "Field Simulations/Data/FarField8AngleDependence.txt"
Data = np.loadtxt(filename)

DataSet = np.zeros(Data.shape)
DataSet[:, 0] = np.abs(np.arctan(Data[:, 1]/Data[:, 0]))
DataSet[:, 1] = Data[:, 2]
DataSet[:, 2] = (np.sqrt(Data[:, 0]**2+Data[:, 1]**2)-1.5)/1.5
DataSet[:, 2] *= Data[:, 2]*170
DataSet[np.where(DataSet[:, 1]<=0), 0] += (np.pi/2)
sigma = np.average(DataSet[:, 2])

##############################################################################

guess = [4, 2, 0, 0]
xtest = np.linspace(min(DataSet[:, 0]), max(DataSet[:, 0]), num=1000)

plt.errorbar(DataSet[:, 0], DataSet[:, 1], yerr=DataSet[:, 2], linestyle='',
             marker='.')
plt.title("Initial Plot of the Quadrupole Angle Dependence Far Field")
plt.xlabel("Radius (meters)")
plt.ylabel("Potential (Volts)")
plt.plot(xtest, farThetaFunc(xtest, *guess))
plt.show()

##############################################################################

Fitparams, Fitcov = cf(farThetaFunc, DataSet[:, 0], DataSet[:, 1], p0=guess,
                       sigma=DataSet[:, 2])
ytest = farThetaFunc(xtest, *Fitparams)

plt.errorbar(DataSet[:, 0], DataSet[:, 1], yerr=DataSet[:, 2], linestyle='',
             marker='.')
plt.title("Fitted Plot of the Quadrupole Angle Dependence Far Field")
plt.xlabel("Radius (meters)")
plt.ylabel("Potential (Volts)")
plt.plot(xtest, ytest)
plt.show()

chi = chiSquared(DataSet[:, 0], DataSet[:, 1], DataSet[:, 2], Fitparams)
dof = len(DataSet[:, 0])-len(Fitparams)
print ("\nGoodness of fit - Chi Square Measure:")
print ("Chi2 = {}, Chi2/dof = {}\n".format(chi, chi/dof))

Fitcov = Fitcov*dof/chi
paramerr = np.sqrt(np.diag(Fitcov))

param_names = ['Coefficient', 'Phase', 'Shift', 'Offset']
```

CurveFitFarQuadrupoleTheta.py(3.3 KB)

Ryan Kaufmann Mar 23, 2018 @12:01 AM PDT

$$V(\theta) = A \sin(B\theta + C) + D$$

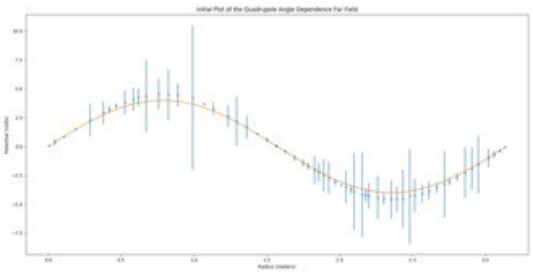
Ryan Kaufmann Mar 23, 2018 @12:03 AM PDT

Pushing our data into our script gives us the following parameters and graphs:

Ryan Kaufmann Mar 23, 2018 @12:05 AM PDT

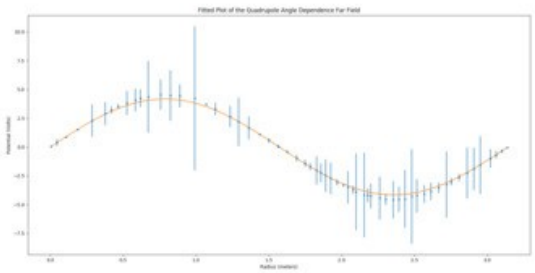
$$A = 4.160536 \pm 0.007684$$
$$B = 2.001772 \pm 0.001390$$
$$C = -0.002958 \pm 0.001256$$
$$D = 0.001074 \pm 0.005068$$

Ryan Kaufmann Mar 23, 2018 @12:08 AM PDT



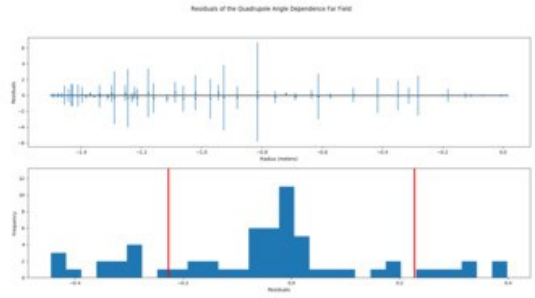
ThetaInitial.png(51.1 KB)

Ryan Kaufmann Mar 23, 2018 @12:08 AM PDT



ThetaFitted.png(51.2 KB)

Ryan Kaufmann Mar 23, 2018 @12:08 AM PDT



ThetaResids.png(38 KB)

Ryan Kaufmann Mar 23, 2018 @12:11 AM PDT

After fitting the function, we get a chi-squared value of 3.316918. However, contrasting this rather high value of chi-squared, we have residuals that both revolve around zero and seems semi-random, or rather no pattern seems to form in them. Thus while our chi-squared does not give a conclusive answer on how well the function has been fitted with the data, our residuals show that it isn't a bad fit. More data points would reinforce the fit function, especially on the second half of the circle. Our final equation for the angle dependent far field becomes:

Ryan Kaufmann Mar 23, 2018 @12:13 AM PDT

$$V(\theta) = 4.160536 \sin(2.001772\theta - 0.002958) + 0.001074$$

Ryan Kaufmann Mar 07, 2018 @06:03 PM PST

Field Simulations 4 - Building a Model

Ryan Kaufmann Mar 23, 2018 @11:53 AM PDT

We can use the above fits to try and make a model for the quadrupole. First, we will start with the easier far field, as it was easier to curve fit. From our numbers above, neither of our equations justify the offset term in the other equation. Thus we can conclude it is less like that the two solutions are simply added together. Instead, we can multiply the equations together in place of a coefficient. That is, our general equation has the following form:

Ryan Kaufmann Mar 23, 2018 @11:53 AM PDT

$$V(r,\theta) = Ar^{B}sin(D\theta + E) + F$$

Ryan Kaufmann Mar 23, 2018 @02:57 PM PDT

We already know the values of B, D, and E, since the multiplication of the equations does not affect these values. However, we should curve fit our data again to test our values of A and F. Using the values we had received from the above equations for B, D, and E. We refit the data with the adapted function, we get the following results from the distance dependence and angle dependence relationship and parameters:

Ryan Kaufmann Mar 23, 2018 @03:02 PM PDT

$$V(r,\theta) = 16.7371 * r^{-3.25985} sin(2.001772\theta - 0.002958) - 0.0047625$$

$$V(r,\theta) = 15.60200 * r^{-3.25985} sin(2.001771\theta - 0.002958) + 0.001073$$

$$A_r = 16.7371 \pm 0.2043$$

$$A_\theta = 15.60200 \pm 0.02882$$

$$F_r = -0.0047625 \pm 0.006018$$

$$F_\theta = 0.001073 \pm 0.005068$$

Ryan Kaufmann Mar 23, 2018 @03:50 PM PDT

Again the data resembles our previous idea that the functions were multiplied against each other. The coefficients are now fairly close to each other and we can estimate it to be the average of the two (16.16955 plus or minus). So, this model of the far field works well, according to the closeness of the coefficients and the chi-squared and residuals.

The near field is harder to model. After several attempts of various fit functions, each ending with either high chi-squared and bad residuals, we settles on the following equation to fit:

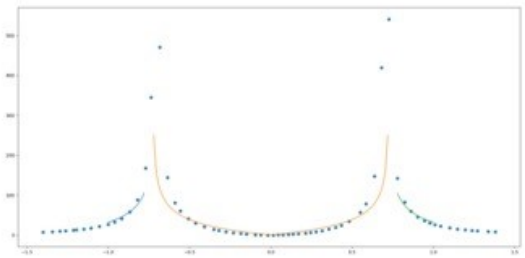
Ryan Kaufmann Mar 23, 2018 @03:55 PM PDT

$$V(r,\theta) = A(\frac{sin(B\theta + C)}{|r - D|^E}) + F$$

Ryan Kaufmann Mar 23, 2018 @04:08 PM PDT

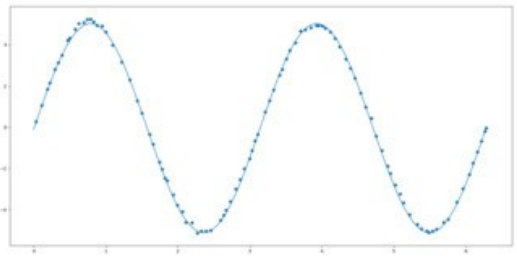
The model doesn't seem to fit well either. We can look at separate parts of the equation and attempt to model first distance then angle. We get the following fits for a constant pi/2 angle and constant 0.25 meter radius:

Ryan Kaufmann Mar 23, 2018 @04:18 PM PDT



NearRadiusFit.png(36.2 KB)

Ryan Kaufmann Mar 23, 2018 @04:18 PM PDT



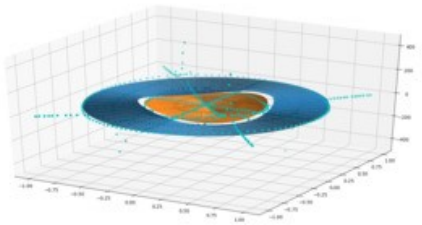
NearThetaFit.png(62.2 KB)

Ryan Kaufmann Mar 23, 2018 @04:49 PM PDT

Although the plots seem good, both fits are not the best. As we can see from the radius plot, as we get closer to the asymptotes, the convergence between the fitted function and the data decreases. That being said, the fit does have a low chi-squared of 0.05. This does not justify the radius function not fitting well to the data. The angle function is not any better. Although the data may seem better, it does not have the same low chi-squared or random residuals. When we run our curve fit, we get a chi-squared of 15332.35. Our fits do not work for each of these. We can conclude that with the data we have, the quadrupole's near field does not follow the same equation as the quadrupole's far field. The area around twice the side length of the quadrupole is a fair marker of the near field/far field division.

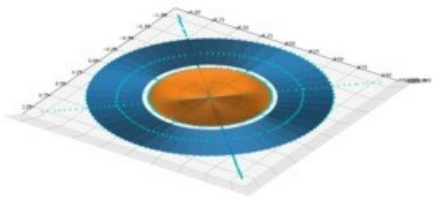
For the sake of at least trying, we also attempted to curve fit the data in both distance and angle dependencies simultaneously using the same equation as above. Our curve fit resulted in the following:

Ryan Kaufmann Mar 23, 2018 @04:49 PM PDT



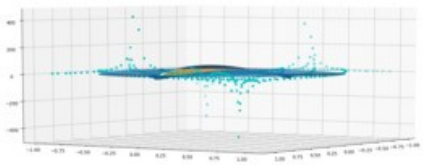
NearFullFit1.png(383.8 KB)

Ryan Kaufmann Mar 23, 2018 @04:49 PM PDT



NearFullFit2.png(523.5 KB)

Ryan Kaufmann Mar 23, 2018 @04:49 PM PDT



NearFullFit3.png(130.1 KB)

Ryan Kaufmann Mar 23, 2018 @04:50 PM PDT

Which again doesn't show a good fit.

Ryan Kaufmann Mar 22, 2018 @06:30 PM PDT

Field Simulations 5 - Conclusion

Ryan Kaufmann Mar 23, 2018 @04:58 PM PDT

Our definitions of the far and near field guides us to make proper approximations in the data. In creating these restrictions, we were able to measure data sets that gave us a reasonable fit for the far field and show us how complex the near field is. In the end, we were able to properly create a model that fully describes the far field for both variations in angle and distance. The near field proved more challenging, and any model we attempted to make was accompanied by either extremely high chi-squared values or patterned residuals, showing us our model did not match our data.

The lab proved useful in learning how to manipulate simulations especially in ways that help reduce uncertainty as much as possible. It was also a good exercise on curve fitting functions that aren't necessarily obvious. Finally, it gave insight into how complicated functions can be and how involved a method of curve fitting can be.