impedance-analysis

May 5, 2019

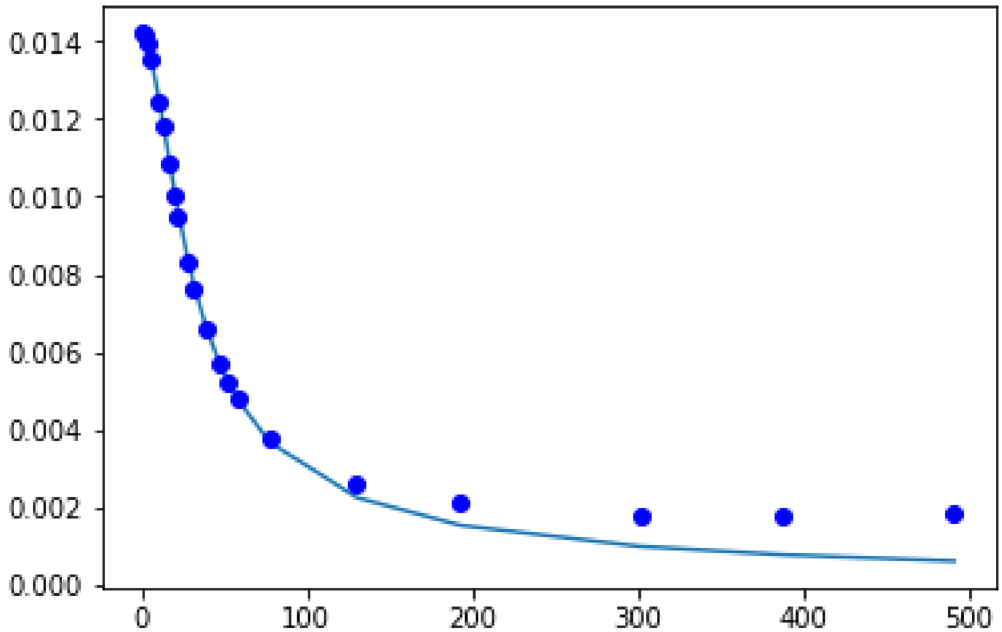
In [1]:
```
# Imports

import numpy as np
from scipy.stats import chisquare
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

import lmfit
#Model = lmfit.Model
from lmfit import Model

pi = np.pi
```

In [2]:
```
# frequency
fk = [96, 268, 630, 995, 1720, 2114, 2665, 3193, 3570, 4460, 5047, 6195, 7531, 8357, 92
f = [i/1000 for i in fk]
#f = np.asarray(f)
w = [2*pi*i for i in f]

# voltage in
v_in = 0.707

# voltage out
v = [14.22, 14.16, 13.95, 13.56, 12.47, 11.81, 10.88, 10.05, 9.48, 8.32, 7.65, 6.60, 5
v = [i*(10**-3) for i in v]

# phase
phi = [-1.3,-3.8,-9.9,-15.1,-24.3,-28.7,-33.8,-37.8,-40.2,-44.3,-46.3,-48.5,-49.8,-49.8
phi = [2*pi*i/360 for i in phi]
phi_inv = [-i for i in phi]
```

In [3]:
```
## Voltage vs. Frequency

# fit function for Vout

def func(x,r0,r1,r2,c):
    num = 1+(x**2)*(c**2)*(r2)*((r1+r2)**2)+(x*c)**2
```

```python
        num = num**(1/2)
        den = 1 + (x**2)*(c**2)*((r1+r2)**2)
        frac = num/den
        fuck = (r1/r0)*frac
        fit = 0.707*fuck
        return fit

    # john's fit

    x = w
    y = v

    model = Model(func)
    model.param_names
    model.independent_vars

    params = model.make_params(r0=10000,r1=200,r2=20.0,c=0.000)
    result = model.fit(y, params, x=x)
    plt.plot(x,result.best_fit)
    plt.plot(x,y,'bo')
    print(result.fit_report())
```

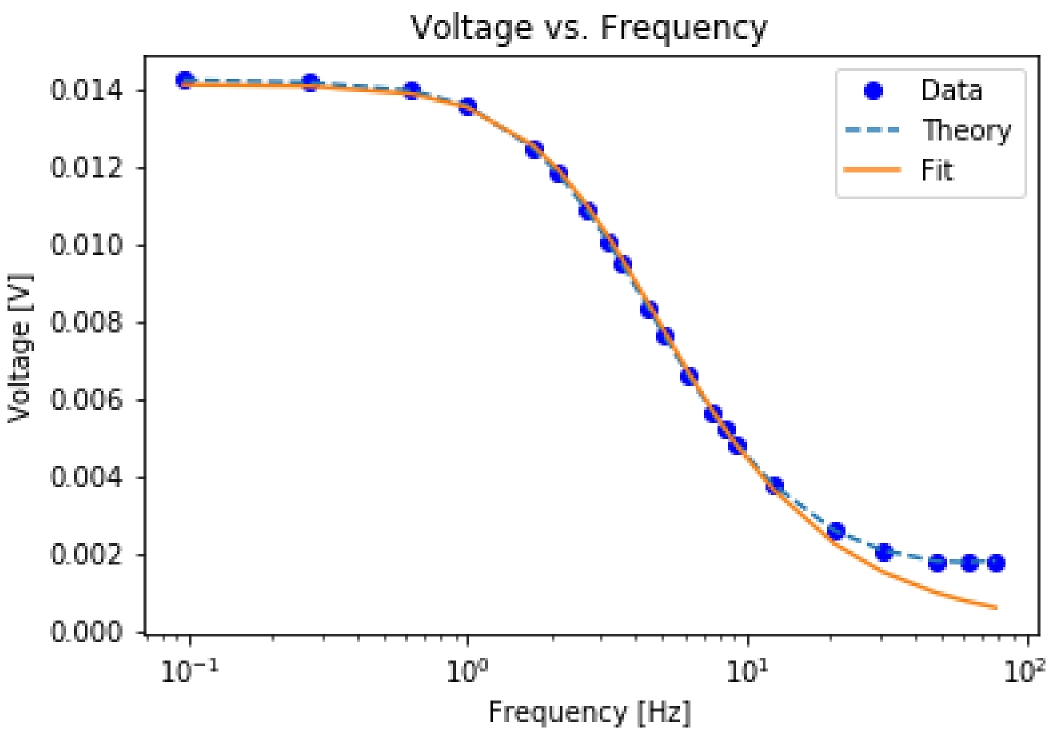
```
[[Model]]
    Model(func)
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 80
    # data points      = 21
    # variables        = 4
    chi-square         = 3.7436e-06
    reduced chi-square = 2.2021e-07
    Akaike info crit   = -318.339719
    Bayesian info crit = -314.161629
[[Variables]]
    r0:  745158.640 +/- 3.3264e+11 (44640630.19%) (init = 10000)
    r1:  14871.9001 +/- 6.6389e+09 (44640628.58%) (init = 200)
    r2:  1.00013281 +/- 1300.60041 (130042.77%) (init = 20)
    c:  -3.2127e-06 +/- 1.43350834 (44619766.92%) (init = 0)
[[Correlations]] (unreported correlations are < 0.100)
    C(r0, r1) =  1.000
    C(r1, c)  =  1.000
    C(r0, c)  =  1.000
    C(r0, r2) =  0.275
    C(r1, r2) =  0.275
    C(r2, c)  =  0.274
```




```
In [4]: # glass fit

        xdata = f
        ydata = v
        p0 = (10020,197.7,20.0,0.000238) #inital guesses based on measured values
        popt, pcov = curve_fit(func,xdata,ydata,p0)
        print(popt)


        # plotting V vs. freq

        x = f
        y = [func(i,popt[0],popt[1],popt[2],popt[3]) for i in f]

        fig, sp = plt.subplots()
        plt.title('Voltage vs. Frequency')
        sp.plot(f,v,'bo',label='Data')
        sp.plot(f,v,'--',label='Theory') #uncomment to view theory
        #sp.plot(w,v,'r--',label='Theory (Angular)') #uncomment to check angular vs linear
        sp.plot(x,y,'-',label='Fit') #uncomment when curve fit is fixed.
        legend = sp.legend(loc='upper right')
        plt.xlabel('Frequency [Hz]')
        plt.ylabel('Voltage [V]')
        plt.xscale('log')
        plt.show()
```

```
K:\Anaconda\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: invalid value encountere
  import sys
```

```
[1.65805060e+04 3.30913109e+02 9.99977271e-01 9.04459906e-04]
```



In [5]:
```
#Create Curve Fit

def func(f,R0f,R1f,R2f,Cf):
    return np.arctan(-2*np.pi*f*Cf/(1+(2*np.pi*f)**2*Cf**2*(R1f+R2f)**2))

popt, pcov = curve_fit(func,f,P)

#Plotting - Phase vs Frequency

fig, sp = plt.subplots()

plt.title('Phase vs. Frequency')
sp.plot(f,P,'bo',label='Data')
#sp.plot(f0,Ph,'--',label='Theory') #uncomment to view theory
#sp.plot(w0,Ph,'r--',label='Theory (Angular)') #uncomment to check for angular vs line
#sp.plot(f,func(f,*parameter),'-',label='Fit') #uncomment when curve fit is working
legend = sp.legend(loc='upper left')
```

```
plt.xlabel('Frequency [Hz]')
plt.ylabel('Phase (Radians)')
plt.xscale('log')

plt.show()
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-5-1f9e4ad0e68a> in <module>
      4     return np.arctan(-2*np.pi*f*Cf/(1+(2*np.pi*f)**2*Cf**2*(R1f+R2f)**2))
      5
----> 6 popt, pcov = curve_fit(func,f,P)
      7
      8 #Plotting - Phase vs Frequency


NameError: name 'P' is not defined
```

In [6]:
```
#Running Loops for Expected values - we will need these for chi squared.
#These are based on theory, not on curve fitting.
#Actually don't do this

#Real(Z)
ReZ_exp = []
for n in f:
    ReZ_exp.append(R1*(1+(2*np.pi*n)**2*C**2*R2*(R1+R2))/(1+(2*np.pi*n)**2*C**2*(R1+R2

#Imaginary(Z)
ImZ_exp = []
for n in f:
    ImZ_exp.append(-R1*(2*np.pi*n)*C*R1/(1+(2*np.pi*n)**2*C**2*(R1+R2)**2))

#Phase
P_exp = []
i = 0
while i < len(ImZ_exp):
    P_exp.append(np.arctan(ImZ_exp[i]/ReZ_exp[i]))
    i += 1

#|Z|
i = 0
MagZ_exp = []
while i < len(ImZ_exp):
    MagZ_exp.append((ReZ_exp[i]**2+ImZ_exp[i]**2)**(1/2))
```

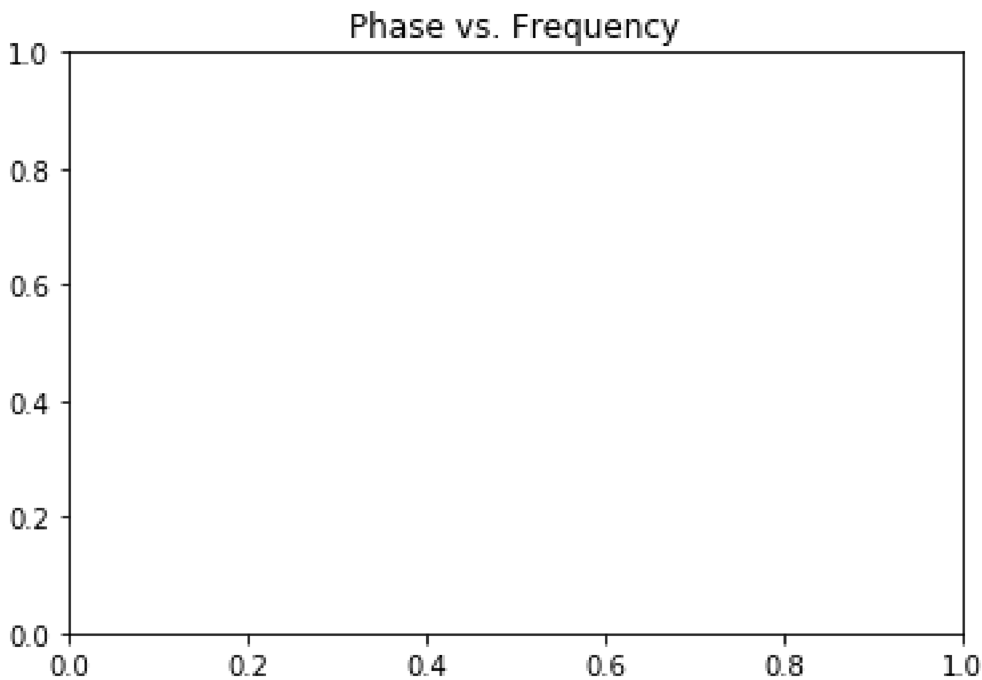
```
    i += 1

#Voltage
V_exp = []
for n in MagZ_exp:
    V_exp.append(I0*n)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-6-6f5801165dc6> in <module>
  5 ReZ_exp = []
  6 for n in f:
----> 7     ReZ_exp.append(R1*(1+(2*np.pi*n)**2*C**2*R2*(R1+R2))/(1+(2*np.pi*n)**2*C**2*(R
  8
  9 #Imaginary(Z)


NameError: name 'R1' is not defined
```

In [7]:
```
#Plotting - Phase vs Frequency

fig, sp = plt.subplots()

plt.title('Phase vs. Frequency')
sp.plot(f,P,'bo',label='Data')
sp.plot(f0,Ph,'--',label='Theory')
#sp.plot(f,P_exp) #checking that expected value calculations are correct.
legend = sp.legend(loc='upper left')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Phase (Radians)')
plt.xscale('log')

plt.show()

#standard deviation calculation

difP = []
i = 0
while i < len(P):
    difP.append((P[i]-P_exp[i])**2)
    i += 1

sP = 0
for n in difP:
```

```
    sP += n

print('Standard Deviation:')
print(sP/len(difP))
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-7-df98c3394f3e> in <module>
      4
      5 plt.title('Phase vs. Frequency')
----> 6 sp.plot(f,P,'bo',label='Data')
      7 sp.plot(f0,Ph,'--',label='Theory')
      8 #sp.plot(f,P_exp) #checking that expected value calculations are correct.


NameError: name 'P' is not defined
```



In [ ]:
```
#plotting - Voltage vs Frequency

fig, sp = plt.subplots()
```

```python
plt.title('Voltage vs. Frequency')
sp.plot(f,V,'bo',label='Data')
sp.plot(f0,V0,'--',label='Theory')
#sp.plot(f,V_exp) #checking that expected value calculations are correct
legend = sp.legend(loc='upper right')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Voltage [V]')
plt.xscale('log')

plt.show()

chisquare(V,V_exp)

#standard deviation calculation

difV = []
i = 0
while i < len(V):
    difV.append((V[i]-V_exp[i])**2)
    i += 1

sV = 0
for n in difV:
    sV += n

print('Standard Deviation:')
print(sV/len(V))
```