Supplementary Information

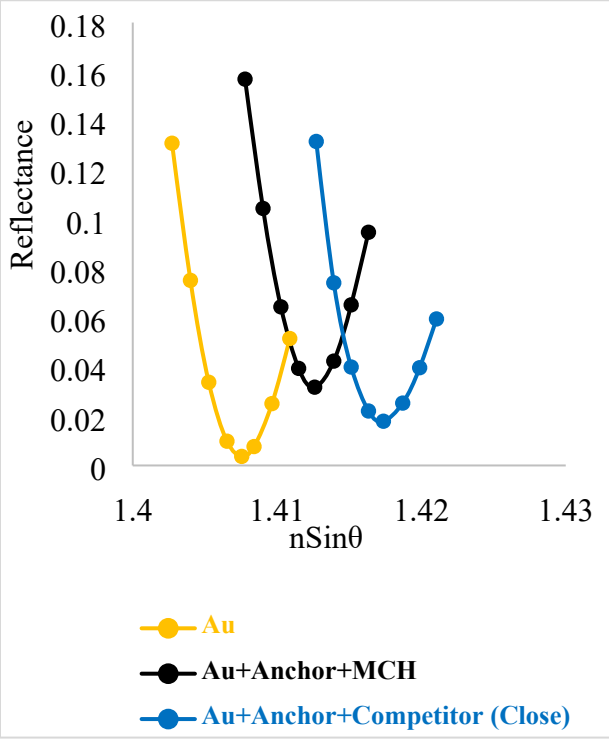
Electrochemically Modulated Surface Plasmon Waves for Characterization and Interrogation of DNA-based Sensors

ANIL SHARMA,[a] THOMAS HULSE,[a] AYMEN H. QATAMIN,[a,b] MONICA MORENO,[a,c] KLESTER S. SOUZA,[a,d] MARCELO B. PEREIRA,[e] FABRICIO S. CAMPOS,[f] LEANDRO B. CARNEIRO,[d] A.M.H. de ANDRADE,[e] PAULO M. ROEHE,[f] FLAVIO HOROWITZ,[e] and SERGIO B. MENDES[a,*]

[a]Department of Physics and Astronomy, University of Louisville, Louisville, Kentucky 40208, USA
[b]Department of Applied Physics, Tafila Technical University, Tafila 66110, Jordan
[c]Department of Chemistry, University of Cauca, Popayan Cl 5 #4-70, Colombia
[d] Institute of Chemistry, Federal University of Rio Grande do Sul, Porto Alegre – RS, 90010-150, Brazil
[e] Institute of Physics, Federal University of Rio Grande do Sul, Porto Alegre – RS, 90010-150, Brazil
[f] Department of Microbiology, Immunology, and Parasitology, Federal University of Rio Grande do Sul, Porto Alegre – RS, 90 050 170, Brazil

** Corresponding author: sbmend01@louisville.edu*

SPR curves



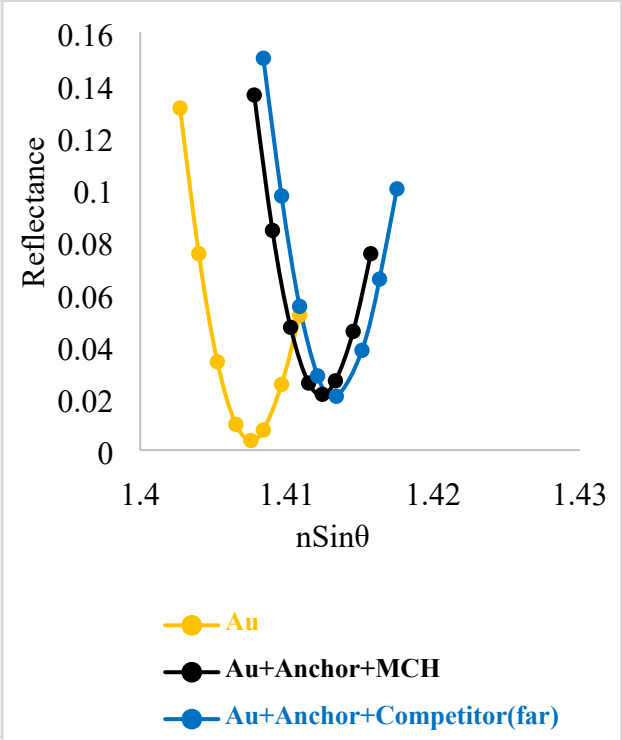


Figure S1. SPR curve for (i) bare gold in PBS buffer (shown in yellow), (ii) anchor primer immobilized on gold surface (shown in black), and (iii) competitor primer hybridized with the anchor primer (shown in blue). On the left, the competitor primer has MB attached on the proximal end while on the right the MB was attached on the distal end of the competitor primer.

Cyclic Voltammograms

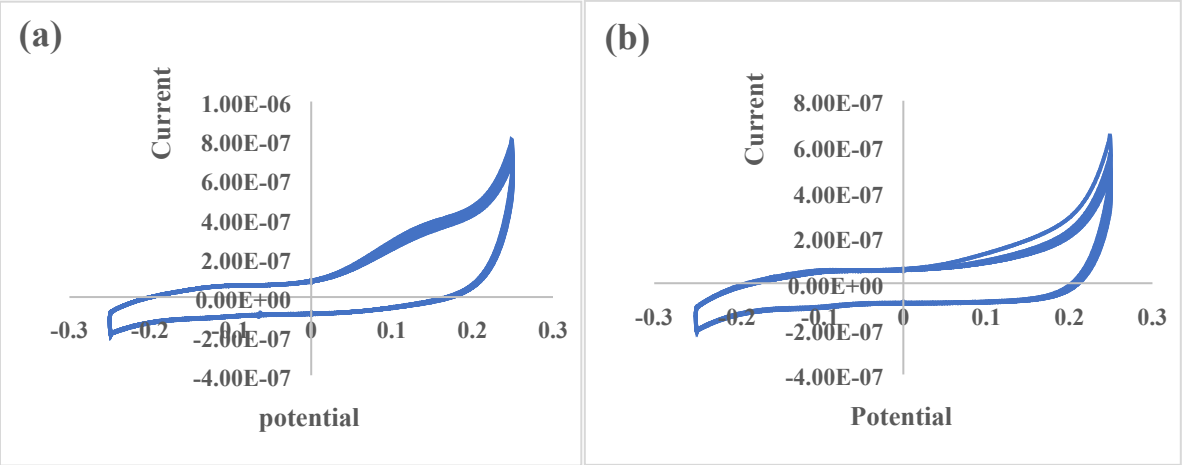


Figure S2. Cyclic voltammograms for 650 nM of competitor primer solution (a) and 650 nM of free MB solution (b). CV scans were taken in the range -0.25 V to 0.25 V at a scan rate of 20 mV/s.

Optimization of Redox Signal

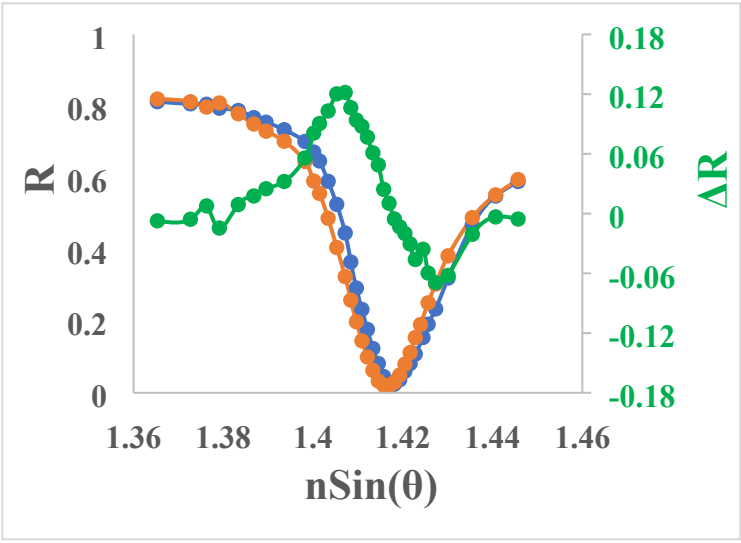


Figure S3. SPR curve for competitor primer hybridized with anchor primer when the potential applied at the working electrode is +0.2 V (blue) and -0.2 V (orange). The difference in reflectance, ΔR (green) is maximum at the left of SPR angle (angle of minimum reflectance)

Reflected Optical Intensity while Taking CV Scan

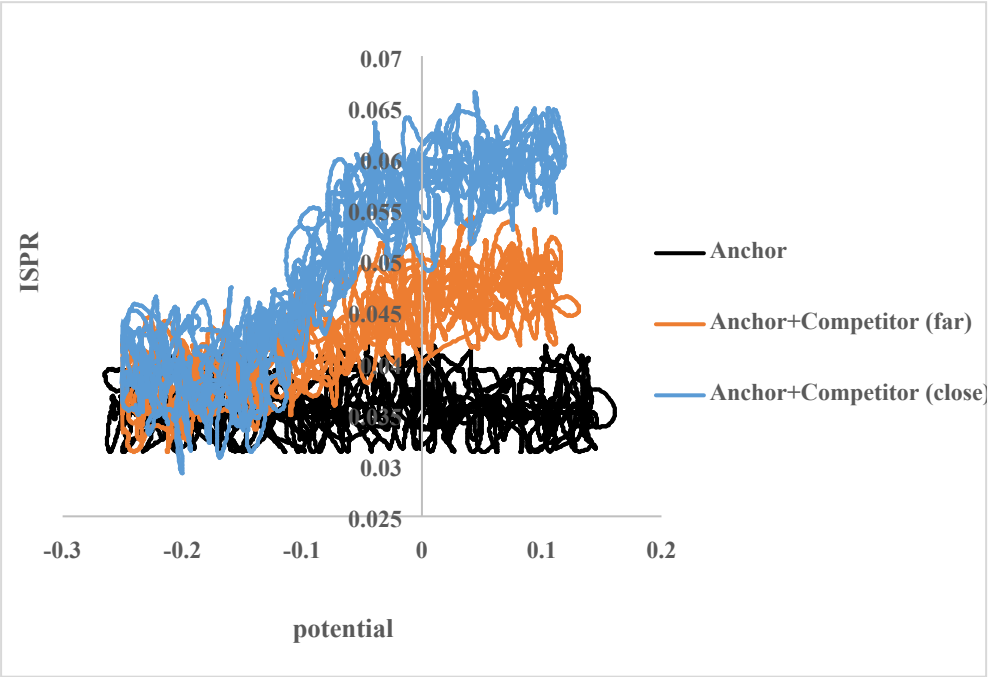


Figure S4. Reflected optical intensity when taking the CV scan for three systems: anchor (shown in black), anchor hybridized with the competitor with methylene blue away from the surface (in orange), and anchor hybridized with the competitor with methylene blue close to the surface (in blue).

Kramers-Kronig Compliance for EIS measurement

With electrochemical impedance spectroscopy, the real $Z^{'}$ and imaginary $Z^{''}$ impedance response of any system satisfying causality, linearity, and stability should obey the following Kramers-Kronig relations:

$$Z^{''}(\omega) = -\frac{2\omega}{\pi}\int_0^{\infty}\frac{Z^{'}(x) - Z'(\omega)}{x^2 - \omega^2}dx \qquad (1)$$

$$Z^{'}(\omega) = Z^{'}(\infty) + \frac{2}{\pi}\int_0^{\infty}\frac{x\,Z^{''}(x) - \omega\,Z''(\omega)}{x^2 - \omega^2}dx \qquad (2)$$

Therefore, in an appropriate system, the raw data for the imaginary component of EIS can be used to generate the real part and vice versa. If the data is reliable, then the integrations should fit well to the raw data, complying with the relations. For example, Kramers-Kronig compliance is shown in Figure 3. The lines each represent the real and imaginary parts generated by the relations in Equations (2) and (3)

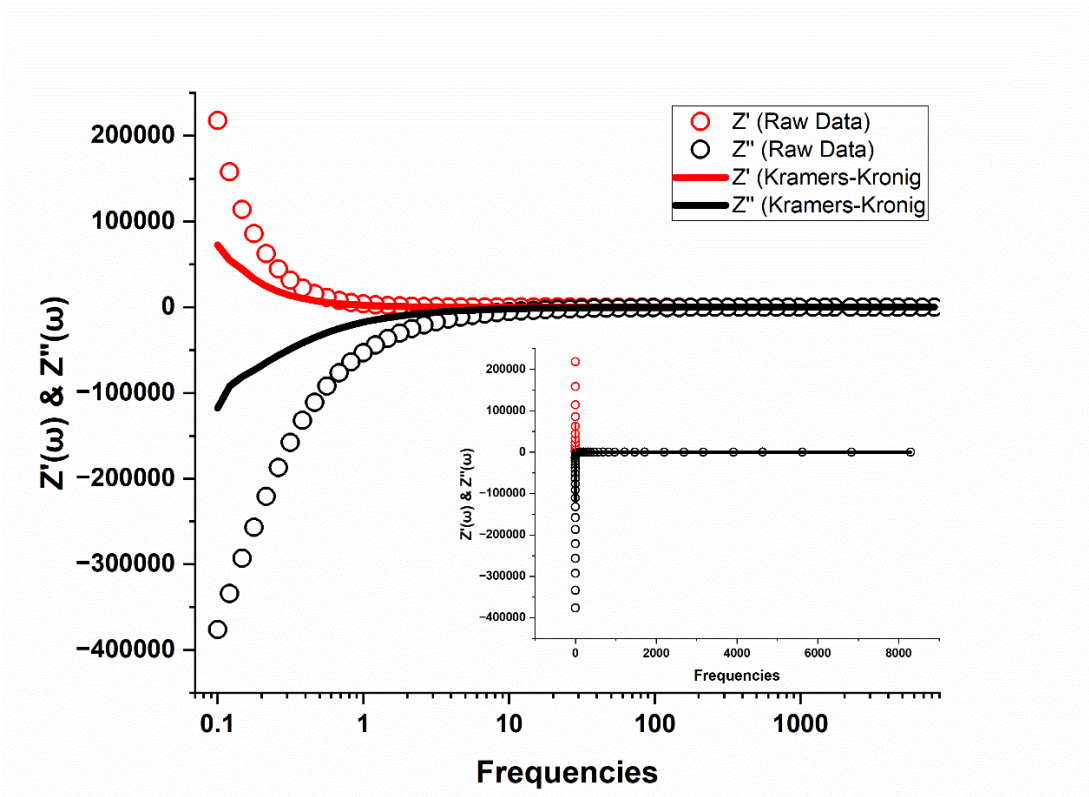


Figure S5: Kramers-Kronig compliance for anchor sample. The open red circles are the raw data for the real component of Z while the open black circles are the raw data for the imaginary component of Z. The red solid line is the real part generated by Equation (3) and the black solid line is the imaginary part generated by Equation (2). The frequency axis is in log scale while the inset is plotted in linear scale.

The integral was numerically evaluated by Simpson's rule using only the raw data. Although the KK-relation tracks relatively well with the raw data, there is some deviation at the low frequency region. This can be primarily attributed to the lack of raw data at low frequencies. The problem of missing frequencies when evaluating the Kramers-Kronig relations is well-known, and despite the missing frequencies, the data here complies reasonably well with the relations. On the other hand, the logarithmic scale can magnify variations at lower frequencies, thereby rendering subtle changes more noticeable, which might remain less obvious when viewed on a linear scale. In the inset of Figure 3, it is evident that the raw data closely matches the KK-relation across the frequency range when plotted on a linear scale.

Python Code for Kramers-Kronig and EIS analysis

```
[1]  # This program takes raw experimental impedance data and plots the numerical Kramers-
     Kronig relations.
[2]  # It then fits the data to a circuit model to find solution resistance, charge transfer resistance,
     double layer capacitance, and pseudo-capacitance.
[3]  # To use, simply adjust the file location in the first non-commented line after the preamble.
[4]  # preamble
[5]  import csv
[6]  import numpy, scipy
[7]  import math
[8]  import matplotlib.pyplot as plt
[9]  from scipy.optimize import curve_fit
[10]
[11] # read from file and process data
[12] file = open('C:/Users/Thomas Hulse/py_projects/KK-Compliance/AC impedance_anchor.csv',
     "r") # adjust file name as needed
[13] data = list(csv.reader(file, delimiter=","))
[14] file.close()
[15]
[16] # find line where data starts
[17] # always after three blank lines, the impedance data begins
[18] i = 0
[19] j = 0
[20] while i < 50:
[21]     if(data[i][0]==''):
[22]         j += 1
[23]         if(j == 3):
[24]             break
[25]     i += 1
[26]
[27] # create arrays for frequencies and real/imaginary impedances. also calculate total impedance
     and phase
[28] frequencies = numpy.array([float(row[0]) for row in data[i+1:]])
[29] ZReal = numpy.array([float(row[1]) for row in data[i+1:]])
[30] ZImag = numpy.array([float(row[2]) for row in data[i+1:]])
[31] totalImp = numpy.sqrt(ZReal**2 + ZImag**2)
[32] phase = -360/2/math.pi*numpy.arctan(ZImag/ZReal)
[33]
[34] # create the imaginary part from real part by KK-relations
```

```
[35] KKImag = [0]*len(frequencies)
[36] integrandArray = [0]*len(frequencies)
[37] integrandDenom = [0]*len(frequencies)
[38]
[39] # compute imaginary component of Z using Simpson's Rule for KK-relations
[40] i = 0
[41] # for each data point...
[42] while i < len(frequencies):
[43]     # compute the integrand for the KK-relations at every point, avoiding the singularity
[44]     integrandDenom = frequencies**2-frequencies[i]**2
[45]     integrandDenom[i] = 1
[46]     integrandArray = (ZReal - ZReal[i])/(integrandDenom)
[47]
[48]     # avoid the singularity
[49]     integrandArray[i] = 0
[50]
[51]     # use Simpson's rule to numerically evaluate integral
[52]     KKImag[i] = -2*(frequencies[i])/numpy.pi*scipy.integrate.simpson(integrandArray,
       frequencies, .1)
[53]     i += 1
[54]
[55]
[56]
[57]
[58]
[59] # create the real part from imaginary part by KK-relations
[60] KKReal = [0]*len(frequencies)
[61] integrandArray = [0]*len(frequencies)
[62]
[63] # compute real component of Z using Simpson's Rule for KK-relations
[64] i = 0
[65] # for each data point...
[66] while i < len(frequencies):
[67]     # compute the integrand for the KK-relations at every point, avoiding the singularity
[68]     integrandDenom = frequencies**2-frequencies[i]**2
[69]     integrandDenom[i] = 1
[70]     integrandArray = (frequencies*ZImag - frequencies[i]*ZImag[i])/(integrandDenom)
[71]
[72]     # avoid the singularity
[73]     integrandArray[i] = 0
```

```
[74]
[75]    # use Simpson's rule to numerically evaluate integral
[76]    KKReal[i] = 2/numpy.pi*scipy.integrate.simpson(integrandArray, frequencies, .1) +
        ZReal[0]
[77]    i += 1
[78]
[79]
[80]
[81]
[82]
[83] # plot (being careful to convert to angular frequency)
[84] plt.xlabel("log(\u03C9)")
[85] plt.ylabel("Z'(\u03C9) & Z"(\u03C9)")
[86]
[87] # scatter + plot the real parts
[88] plt.scatter(numpy.log10(2*numpy.pi*frequencies), ZReal, label='Z\' Raw Data', s = 20)
[89] plt.plot(numpy.log10(2*numpy.pi*frequencies), KKReal, color='red', label = 'Z\' Kramers-
        Kronig', linewidth = 2.5)
[90]
[91] # scatter + plot the imaginary parts
[92] plt.scatter(numpy.log10(2*numpy.pi*frequencies), ZImag, label='Z\'\' Raw Data', s = 20)
[93] plt.plot(numpy.log10(2*numpy.pi*frequencies), KKImag, color='black', label = 'Z\'\'
        Kramers-Kronig', linewidth = 2.5)
[94]
[95] # legend + show
[96] plt.legend(loc="center right")
[97] plt.show()
[98]
[99] # define various circuit models for real impedance, imaginary impedance, total impedance,
        and phase
[100]     # note: R1 is solution resistance, R2 is charge transfer resistance, C1 is double layer
        capacitance, C2 is pseudo-capacitance
[101]
[102]     # shorthand for frequencies
[103]     x = frequencies
[104]
[105]
[106]
[107]
[108]
```

```
[109]     # model for real part
[110]     def modReal(data, R1, R2, C1, C2):
[111]         R1 = ZReal[0]
[112]         # analytical solution of circuit
[113]         result = R1 + (-R2/(C1*C2*((2*math.pi*x)**2)) + R2*(1/((2*math.pi*x)*C1) +
          1/((2*math.pi*x)*C2))/((2*math.pi*x)*C1))/(R2**2 + (1/((2*math.pi*x)*C1) +
          1/((2*math.pi*x)*C2))**2)
[114]         # force all parameters to be positive and reasonable (no massive capacitances)
[115]         if(R1 > 0 and R2 > 0 and 1 > C1 > 0 and 1 > C2 > 0):
[116]             return result
[117]         else:
[118]             return 1e10
[119]
[120]
[121]
[122]
[123]
[124]     # model for imaginary part
[125]     def modImag(data, R1, R2, C1, C2):
[126]         R1 = ZReal[0]
[127]         # analytical solution of circuit
[128]         result = -(R2**2/((2*math.pi*x)*C1) + (1/((2*math.pi*x)*C1) +
          1/((2*math.pi*x)*C2))/(C1*C2*(2*math.pi*x)**2))/(R2**2 + (1/((2*math.pi*x)*C1) +
          1/((2*math.pi*x)*C2))**2)
[129]         # force all parameters to be positive and reasonable (no massive capacitances)
[130]         if(R1 > 0 and R2 > 0 and 1 > C1 > 0 and 1 > C2 > 0):
[131]             return result
[132]         else:
[133]             return 1e10
[134]
[135]
[136]
[137]
[138]
[139]     # model for total impedance
[140]     def modTotImp(data, R1, R2, C1, C2):
[141]         R1 = ZReal[0]
[142]         # real and imaginary parts added in quadrature
[143]         result = numpy.sqrt((R1 + (-R2/(C1*C2*((2*math.pi*x)**2)) +
          R2*(1/((2*math.pi*x)*C1) + 1/((2*math.pi*x)*C2))/((2*math.pi*x)*C1))/(R2**2 +
```

```
    (1/((2*math.pi*x)*C1) + 1/((2*math.pi*x)*C2))**2))**2 + (-(R2**2/((2*math.pi*x)*C1) +
    (1/((2*math.pi*x)*C1) + 1/((2*math.pi*x)*C2))/(C1*C2*(2*math.pi*x)**2))/(R2**2 +
    (1/((2*math.pi*x)*C1) + 1/((2*math.pi*x)*C2))**2))**2)
[144]       # force all parameters to be positive and reasonable (no massive capacitances)
[145]       if(R1 > 0 and R2 > 0 and 1 > C1 > 0 and 1 > C2 > 0):
[146]           return result
[147]       else:
[148]           return 1e10
[149]
[150]
[151]
[152]
[153]
[154]   # model for phase in degrees
[155]   def modPhase(data, R1, R2, C1, C2):
[156]       R1 = ZReal[0]
[157]       result = -360/2/math.pi*numpy.arctan(-(R2**2/((2*math.pi*x)*C1) +
    (1/((2*math.pi*x)*C1) + 1/((2*math.pi*x)*C2))/(C1*C2*(2*math.pi*x)**2))/(R2**2 +
    (1/((2*math.pi*x)*C1) + 1/((2*math.pi*x)*C2))**2)/(R1 + (-
    R2/(C1*C2*((2*math.pi*x)**2)) + R2*(1/((2*math.pi*x)*C1) +
    1/((2*math.pi*x)*C2))/((2*math.pi*x)*C1))/(R2**2 + (1/((2*math.pi*x)*C1) +
    1/((2*math.pi*x)*C2))**2)))
[158]       # force all parameters to be positive and reasonable (no massive capacitances)
[159]       if(R1 > 0 and R2 > 0 and 1 > C1 > 0 and 1 > C2 > 0):
[160]           return result
[161]       else:
[162]           return 1e10
[163]
[164]   # fitting phase model
[165]   # initial guess for parameter values
[166]   initialParameters = numpy.array([ZReal[0], 1e5, 2e-6, 1e-6])
[167]
[168]   # find fitted parameters with curve_fit
[169]   fittedParameters, pcov = curve_fit(modPhase, frequencies, phase, initialParameters,
    maxfev=5000)
[170]   R1, R2, C1, C2 = fittedParameters
[171]   print(fittedParameters)
[172]
[173]   y_fit_totImp = modTotImp(frequencies, R1, R2, C1, C2) # third data set, third equation
[174]   y_fit_phase = modPhase(frequencies, R1, R2, C1, C2) #fourth data set, fourth equation
```

```
[175]
[176]     # fitted plots
[177]
[178]     # plotting total impedance vs frequency
[179]     plt.scatter(numpy.log10(frequencies), numpy.log10(y_fit_totImp), label = 'Raw Data', s =
      20)
[180]     plt.plot(numpy.log10(frequencies), numpy.log10(y_fit_totImp), color = 'red', label = 'Fit',
      linewidth = 2.5)
[181]     plt.xlabel("log(\u03C9)")
[182]     plt.ylabel("Z")
[183]     plt.legend(loc="center right")
[184]     plt.show()
[185]
[186]
[187]
[188]
[189]
[190]     # plot phase vs frequency
[191]     plt.scatter(numpy.log(2*numpy.pi*frequencies), phase, label = 'Raw Data', s = 20)
[192]     plt.plot(numpy.log(2*numpy.pi*frequencies), y_fit_phase, color='red', label = 'Fit',
      linewidth = 2.5)
[193]     plt.xlabel("ln(\u03C9)")
[194]     plt.ylabel("\u03f4")
[195]     plt.show()
```