Percolation Threshold Model Summary

Patrick Blah

1 Reason for Model

Some materials, such as $NdNiO_3$, have a strong metal-to-insulator transition (MIT) where this transition can be observed by measuring its electrical resitivity vs temperature. Also hysteretic behaviour can be observed, where an example of an MIT I measured is shown below:

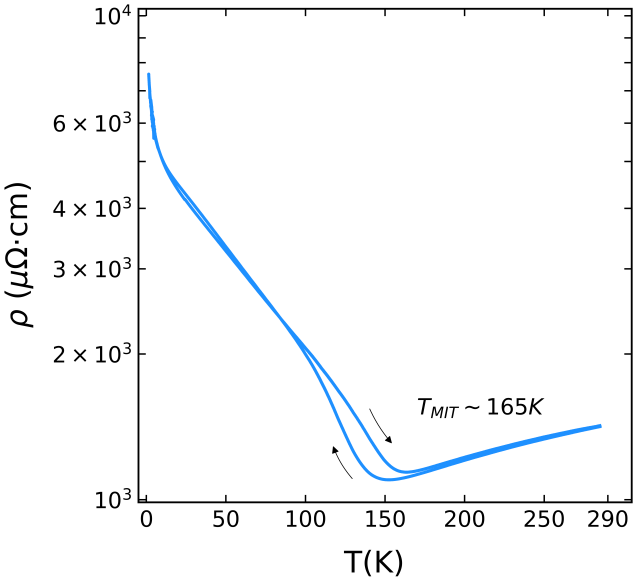

Figure 1: Resitivity vs Temperature of $NdNiO_3$. A clear MIT is observed as well as hysteretic behaviour.

Since there is a drastic change in resitivity as one crosses a temperature threshold (around 165K for this sample), utilising methods which describe critical behaviour is a useful way to describe the phenomena observed. It can be seen that above 165K the material acts as a metal, where the slope of its resitivity vs temperature is linear. Meanwhile below 165K the slope quickly becomes non-linear, with its slope being described better by an exponential. Meanwhile the region between 100K and 165K is unusual as a hysteresis loop is observed. This hysteresis shows that the transition from a metal to an insulator is not abrupt, and that there is an additional energy barrier involved in this transition. One way to model this transition is to imagine that the material is made up of a number of domains, where these domains are either fully metallic or semiconducting.

Consider the following: The material is at room temperature, thus nearly all of the domains are fully metallic. As you cool the material down below 165K a number of these domains become insulating. As you cool down the material further more and more of these domains become semiconducting while at a certain point (around 100K) almost all of the domains are semiconducting. Say then you start to warm up the material from 1.5K, more and more of the domains start to become metallic while above 165K the majority of the domains are metallic again. So one question is, **is there a critical ratio of metallic to semiconducting domains where this MIT occurs?** It turns out that there is indeed a value that is generic across multiple systems and materials, and we can use this to confirm that this material experiences domain-based percolation.

2 Origin of Model

This was investigated around the 1950s where a very nice generic model was produced to describe percolation networks and critical phenomena. The model was constructed via the following:

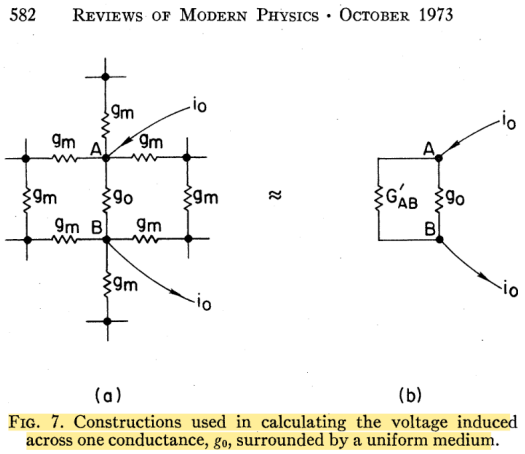

FIG. 7. Constructions used in calculating the voltage induced across one conductance, g_0, surrounded by a uniform medium.

Figure 2: –

Say you have a random resistor network represented by bonds that are either metallic (aka.Ohmic) or semi-conducting. **A key assumption we now make is that this random network can be represented by a homogeneous one where each bond has an 'effective' conductance** g_m. Additionally, we designate one bond as the bond where the electricity actually flows through the network, and that the potential over the rest of the bonds, as well as this special bond, averages out to zero over a sufficiently large region. Essentially, you apply an external potential field, the potential is only considered across this 'special' bond while the other bonds cancel each other out. If you do this then you can consider Figure 2(a), where a homogeneous resistor network with these 'effective' bonds is shown. g_m is the 'effective' conductance as described previously and g_0 is the conductance of this special bond. This special bond connects the points A and B.

What you can do then is basically apply superposition and then Kirchoff's current Law around the circuit loop. But one has to be careful as Kirchoff's current law doesn't hold at points A and B. Thus an additional current i_0 is introduced which starts at A and ends at B. The magnitude of this current is then:

$$i_0 = V_m(g_m - g_0) \tag{1}$$

(Where $\sum_j g_{ij}(V_i - V_j) = 0$ is Kirchoff's current law for a random network at neighbouring points i,j). So we want to know what potential, V_0, causes this current i_0. You can begin to work this out by finding the conductance of the homogeneous network between A and B (so considering all the bonds but the special bond). This is shown by the circuit diagram in Figure 2(b) where this value is G'_{AB}. Doing Ohm's Law around this loop, you get:

$$V_0 = \frac{i_0}{g_0 + G'_{AB}} \tag{2}$$

So plugging Equation 2 into Equation 1 we get:

$$V_0 = \frac{V_m(g_m - g_0)}{g_0 + G'_{AB}} \tag{3}$$

So what's G'_{AB}? It's:

$$G'_{AB} = (\frac{z}{2} - 1)g_m \tag{4}$$

Where z is the number of bonds connected at the point where the current enters the designated bond (ie. the number of bonds at point A in Figure 2(a) which is 4). The $\frac{z}{2}$ comes from the fact that; when working out G'_{AB} you consider the special bond as just a normal bond with conductance g_m. Then, since the current entering point A has to equal the current exiting, and the 4 bonds each have equal conductances, then the current has to flow in via two bonds and exit via 2 bonds. This means that the current that then travels through the homogeneous circuit, while excluding the bond between A and B, is halved. You then include a -1 as that takes into account you not considering the bond between A and B. You then multiply the number of

bonds the current travels through by the conductance of each bond (g_m). Note: z depends on the dimension of the circuit, where z = 4 for 2D and z=6 for 3D. Plugging Equation 4 into Equation 3 you get:

$$V_0 = \frac{V_m(g_m - g_0)}{g_0 + (\frac{z}{2} - 1)g_m} \tag{5}$$

With the voltage developed across the bond AB now described in terms of conductances, it will be important to include the probability what type of bond AB will be when we designate it as the special bond (aka.Ohmic or semiconducting) as well as the fact that we need to satisfy the boundary condition that this special bond averages out over a large enough region. For the probability, the probability distribution of a bond with a conductance g can be described by the following:

$$f(g) = P_M\delta(g - g_M) + P_S\delta(g - g_S) \tag{6}$$

Where P_M,P_S is the probability of the bond being metallic or semiconducting respectively. g_M,g_S are the conductances of a metallic or semiconducting bond, respectively. You can apply this to the previously described boundary condition by:

$$\int f(g)\frac{V_m(g_m - g_0)}{g_0 + (\frac{z}{2} - 1)g_m}dg \tag{7}$$

Where using Equation 6 this becomes:

$$\frac{P_M(g_m - g_M)}{g_M + (\frac{z}{2} - 1)g_m} + \frac{P_S(g_m - g_S)}{g_S + (\frac{z}{2} - 1)g_m} = 0 \tag{8}$$

You can assume that the probability of the bond being semiconducting is equal to the relative volume of the semiconducting region, hence $P_S = V_S$ and $P_M = 1 - V_S$ where V_S is the volume of the semiconducting region. Additionally, it is convenient to use the notation $z = 2D$ where D is the dimension of the system. Applying this and rearranging you get:

$$V_S = \frac{(g_m - g_M)(g_S + g_m(D - 1))}{DG_e(g_s - g_m)} \tag{9}$$

This is the equation that is used when modeling this system.

3 Applying the Model

So say you have the data shown in Figure 1. You need the parameters in Equation 9. g_M is found by performing a linear fit of the data in temperature region well above the metal-insulator transition (aka. 200-300K). g_S is found by performing a fit of the data below the temperature at which the hysteris loop closes (aka. 1.5 - 100K). Starting with g_M:

```
for i,data in enumerate(pathlist_RT_film_cleaned):



    if i==0:



        dataextracted = dataextractorRT(data)
        temperature = dataextracted[0]
        resistance2pt = dataextracted[1]
        resitivity2pt = resistance2pt *22.86E-9*(np.pi/np.log(2))*1E2*1E6
        resistance4pt = dataextracted[3]
        resitivity4pt = resistance4pt *22.86E-9*(np.pi/np.log(2))*1E2*1E6

        CCLXXX = int(closest_element_index(temperature,280)[0])
        CC = int(closest_element_index(temperature,165)[0])

        print(CCLXXX)
        print(CC)

        temperature_metallic_region = temperature[CCLXXX:CC]
        resitivity4pt_metallic_region = resitivity4pt[CCLXXX:CC]
        print(temperature_metallic_region)
        print(resitivity4pt_metallic_region)

        conductivity4pt_metallic_region = 1/(resitivity4pt[CCLXXX:CC])



        plt.plot(temperature_metallic_region,conductivity4pt_metallic_region, color = 'black')
```

Figure 3:

The resistance data is in the form of a txt file, it's extracted and the two types of resistances (performed in 2-point measurements and 4-point measurements) are converted to resitivities via the formula:

$$\rho = R * t * \frac{\pi}{2} \tag{10}$$

where R is the resistance and t is the thickness of the material. The $1E2 * 1E6$ is for unit conversion later. The conductance of the material is the inverse of the resitivity. The temperature region is defined by CCLXX and CC (280 and 200 in Roman numerals) where the function 'closest element index' is used. This function finds the datapoint closest to a defined value:

```
"--- Closest Element Range Function ---"

def closest_element_index(array,value):
    array1 = np.sort(array)
    closest_element = min(array1, key=lambda x:abs(x-value))
    closest_element_index = np.where(array1 == closest_element)[0][0]
    closest_index_range = array1[closest_element_index-1 : closest_element_index+1]
    mylist = []
    for i in closest_index_range:
        closest_index_actual = np.where(array == i)[0]
        mylist = np.sort(np.append(mylist,closest_index_actual))
    return mylist
```

Figure 4:

The data in the temperature range 280-200K is then fitted via the Numpy Polynomial.Polynomial class, where the polynomial is 1st order. The fit parameters are extracted and stored in a pandas Dataframe.

```
##### Numpy Polynomial Fit 1st Order #####

a, b = np.polynomial.polynomial.polyfit(temperature_metallic_region,conductivity4pt_metallic_region, 1)
print('a',a)
print('b',b)
fit1 = a + b*temperature_metallic_region
print('fit1',fit1)
print('Polynomial Fit 1st Order', np.polynomial.polynomial.Polynomial([a,b]))
plt.plot(temperature_metallic_region, fit1, linestyle = "--", linewidth = 2, color = 'orange', alpha = 1)

pd.DataFrame({'a':[a],'b':[b]}).to_csv(r'C:\Users\pblah\Data\Navy Beach\FM318\Data\Film\RT\Fitting Parameters\Linear Fit\ ' +
'Linear_Fitting_Params' + '.csv')

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

Figure 5:

With those parameters, we can now define our g_M parameter as:

```
def g_m(T,a,b):
    g_m = a + b*T
    return g_m
```

Figure 6:

Note, the code notation uses g_m for g_M (apologies!).

Now for g_S. The data in the semiconductor region is set up in the same manner, where the resistance data is imported (the data was already a Dataframe in this case), converted to a numpy array and the resitivity is calculated. The inverse is then calculated to find the conductance. The temperature region is defined by 'h' and 'l' using the 'closest element index' function. The temperature range here was 30-160K. But how to fit the data in this region? It turns out it's best described by two exponentials based on the two conduction mechanisms occurring in the insulating regions (variable range hopping and normal activation energy):

$$\sigma = Ae^{\frac{-B}{T^{\frac{1}{4}}}} + Ce^{\frac{-D}{T}} \tag{11}$$

Where A,B,C,D are constants and T is temperature. The σ in Equation 11 is our g_S parameter:

```
def g_s(T,A,B,C,D):
    g_s = A*np.exp(-B/(T**(1/4))) + C*np.exp(-D/(T))
    return g_s
```

Figure 7:

Scipy 'curve fit" is used, which uses non-linear least squares to fit data via a defined function. The parameters A,B,C and D are extracted and stored in a dictionary.

```
parameters, covariance = curve_fit(combo, temperature_insulating_region, r4pt_inv_insulating_region,maxfev=50000)
fit_A = parameters[0]
fit_B = parameters[1]
fit_C = parameters[2]
fit_D = parameters[3]
print("A,B,C,D",fit_A,fit_B,fit_C,fit_D)
fit_y = combo(temperature_insulating_region, fit_A, fit_B, fit_C, fit_D)

plt.plot(temperature_insulating_region, r4pt_inv_insulating_region, lw = 4, color = colours2[i])
plt.plot(temperature_insulating_region, fit_y, lw = 4, color = colours2[(i+1)*2], linestyle = "--", alpha = 0.5)

parameters_dict['fit' + '_' + 'A' + '_' +  types_membrane[i]] = fit_A
parameters_dict['fit' + '_' + 'B' + '_' +  types_membrane[i]] = fit_B
parameters_dict['fit' + '_' + 'C' + '_' +  types_membrane[i]] = fit_C
parameters_dict['fit' + '_' + 'D' + '_' +  types_membrane[i]] = fit_D
```

Figure 8:

So we now have g_S. The final parameter required, g_m, is simply the actual conductance values measured, hence no further work is required. Finally the volume fraction of the semiconductor regions, V_S, can be plotted versus temperature:

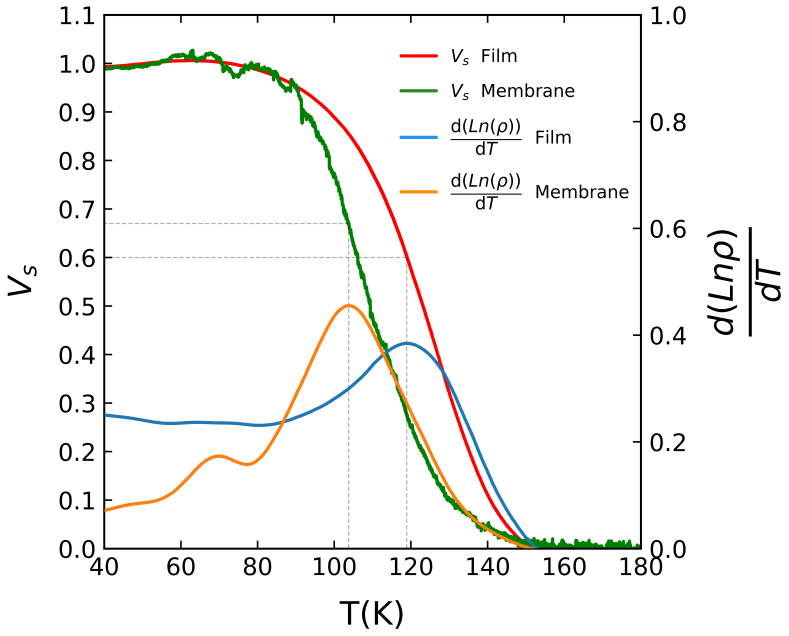

Figure 9:

4 Interpreting the Results

There's a fair amount going on in Figure 9. Ignoring the right vertical axis for now, this plot describes measurements performed on a sample first in its 'Film' form and then in its 'Membrane' form. The purpose of the measurements was to compare the behaviour between the two. The volume fraction of the semiconducting region behaves as we expect, where it is almost 1 far below the metal-insulator-transition temperature (¡80K) and almost 0 far above it (¿160K). The right axis is the derivative of the logarithm the resitivity vs temperature. The reason for calculating and plotting this comes from certain trends found in percolation systems, where it is observed that the point at which one sees the largest degree of change during a transition (aka. the slope of a certain parameter during a critical 1st order transition) almost always corresponds to a volume fraction ~ 0.6. This applies not just to complex oxides (what $NdNiO_3$ is) but also to other percolation systems of this nature as well. The purpose of these measurments in this work was to compare the properties of this metal-insulator transition of $NdNiO_3$ in its film and membrane form. The plot confirms that the material indeed retains its intrinsic properties in its membrane from with a slight increase of the volume fraction at the peak of the right axis. While beyond the scope of this text, this peak corresponds to an increase in certain defects occurring in the material, where defects in general result in more semiconducting regions forming and being pinned.

```
############################ Smooth Finite Difference: Mean smoothing ##########################

params, val = pynumdiff.optimize.smooth_finite_difference.meandiff(Ln_Rho[i], 0.01, params=None,
                                                                   options={'iterate': True},
                                                                   tvgamma=tvgamma,
                                                                   dxdt_truth=None)

x_hat, dxdt_hat = pynumdiff.smooth_finite_difference.meandiff(Ln_Rho[i],0.01, params, options={'iterate': True})
```

Figure 10:

Lastly the right axis, which is a derivative of somewhat noisy data, was created using the package pynumdiff which, when broken down, uses scipy's filter fit with an added first-order finite difference.