Comparing numerical results with experimental data

February 20, 2020

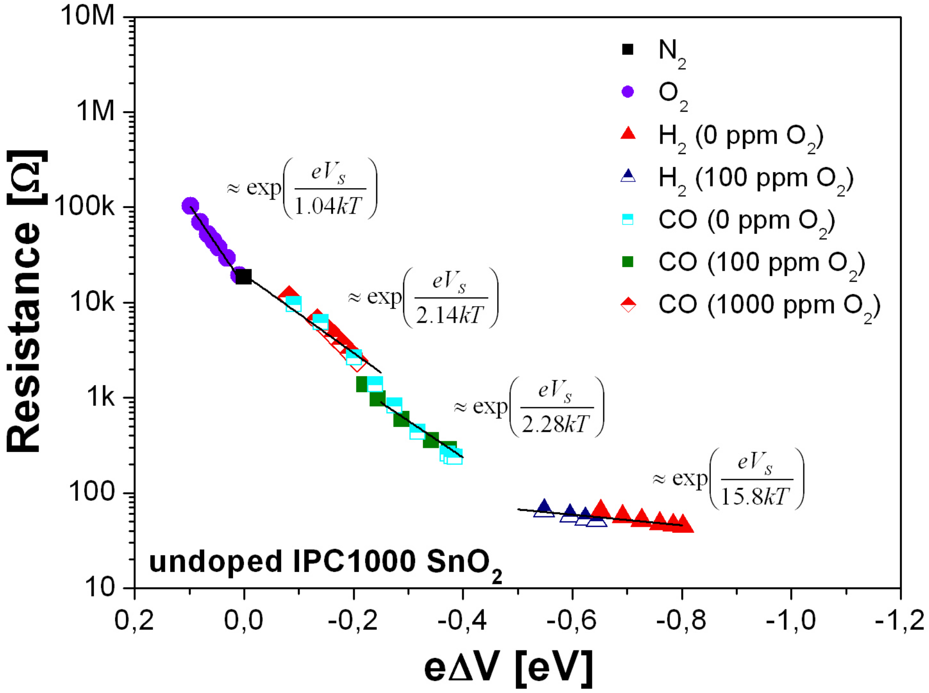
Contents

1 Abstract

Experimental data from simultaneous work function and resistance measurements will be compared with the results from the numerical calculations. Results from an SnO_2 gas sensor measured at 300°C will be used to demonstrate, how numerical data can be used to gain more insights about the measured material. The chosen dataset the the graphical representation originates from the Phd. thesis of Julia Rebholz: [Reb16].

The data was generated by exposing the sensor to various gas compositions of H_2, CO, O_2 and N_2. The surface potential changes ΔV resulting from the different gas atmosphere have been obtained with the Kelvin probe technique. Simultaneously the corresponding resistance was measured. The data point point at $0e\Delta V$ corresponds to the situation in nitrogen.

These experimental data points will be compared with the results obtained from the numerical model.



2 Fitting experimental data to numerical results

2.1 Importing experimental and numerical data

The data is saved in an excel file, which will be loaded by using the tools provided by pandas.

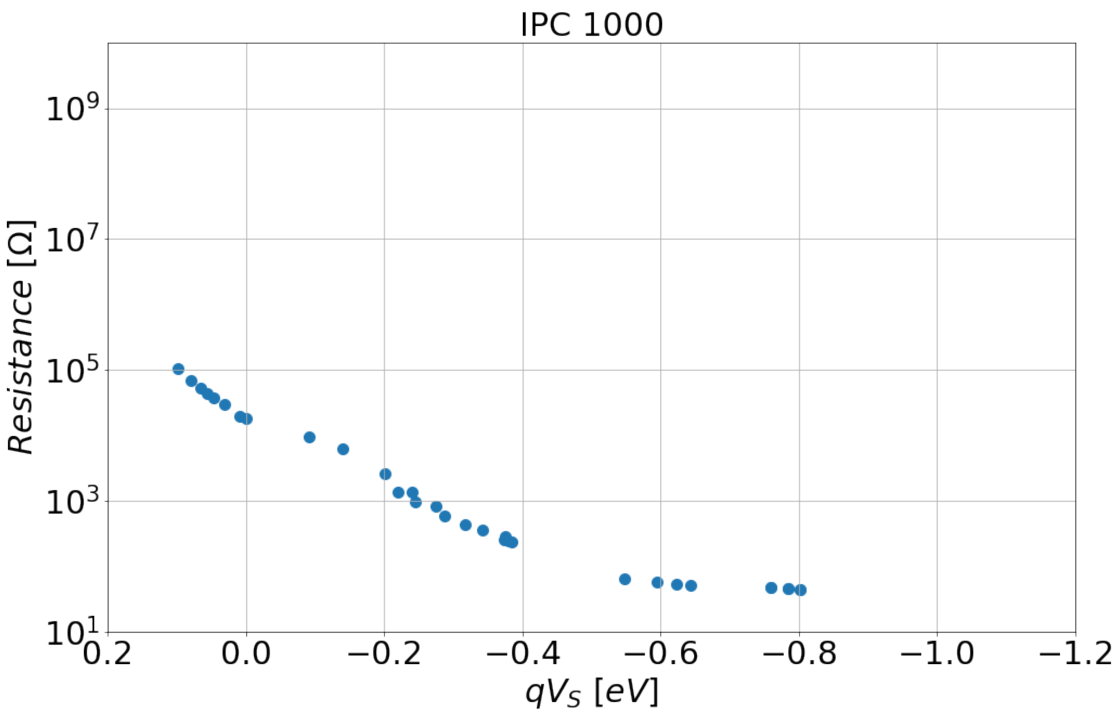
[43]:
```
#Setting up the env.
from part2 import *
import pandas as pd
%pylab inline

#importing the data
calc_dF = pd.read_hdf('numerical_sol.h5','raw')
dF_1000 = pd.read_excel('Kelvin_Data.xlsx', sheet_name='ipc1000').
 ↪sort_values(by='dV')

#instead of unsing the row number
#each row has the value of dV as index
dF_1000.index = dF_1000['dV']
```

```
Populating the interactive namespace from numpy and matplotlib
```

2.2 Representing the raw data

[47]:

```
def format_axe(axe, ylabel = None, set_ylim=False):
    labelsize = 30
    if set_ylim:
        axe.set_ylim((1e-4,1e3))
    axe.set_yscale('log')
    axe.set_xlim((0.2,-1.2))
    if ylabel:
        axe.set_ylabel(ylabel, fontsize = labelsize)
    else:
        axe.set_ylabel(r'$\frac{R_{V_S}}{R_{(V_S=0)}}$', fontsize = labelsize)
    axe.set_xlabel('$qV_S$ $[eV]$', fontsize = labelsize)
    axe.tick_params(axis='both', which='both', labelsize=labelsize)


fig, axe = subplots(figsize=(16,10))

sens, dF = 'IPC 1000',dF_1000

v_exp = dF['dV']

res_exp = dF['res']
axe.set_title(sens, fontsize = 30)
axe.scatter(v_exp,res_exp, s=100)
format_axe(axe,ylabel='$Resistance$ [$\Omega$]')
axe.set_ylim(res_exp.min()/2,res_exp.max()*2);
axe.set_ylim(10,10e9);
axe.grid()
```



2.3 From R_{V_S} to \$\Delta R_{V_S} \$

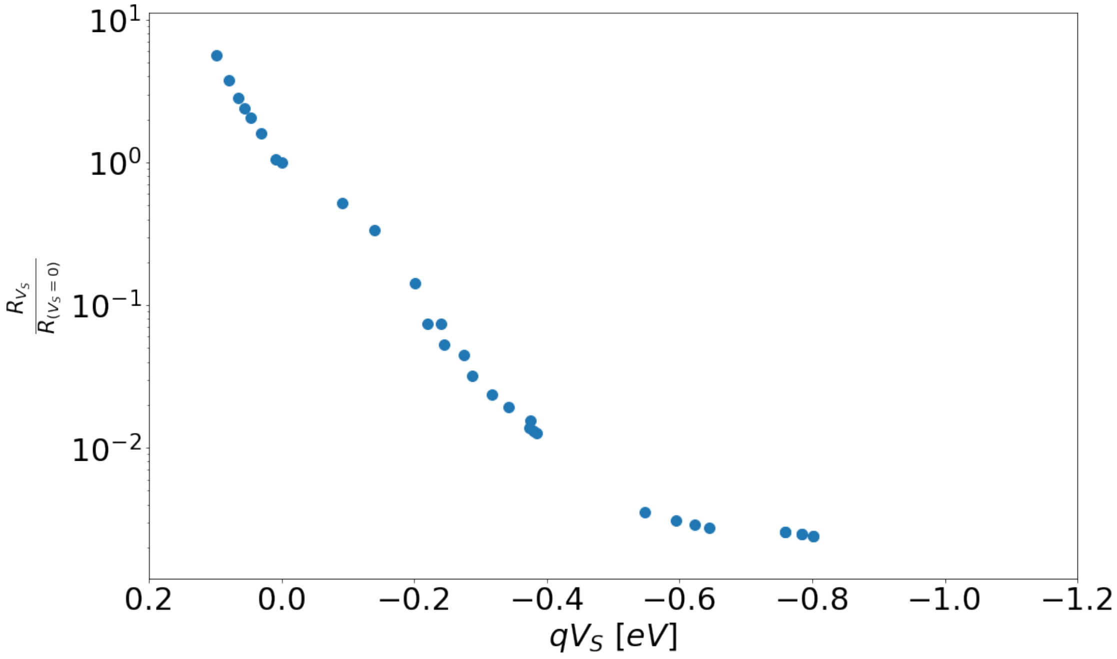
In the experimental dataset the value at $0qV_s$ represent the data points measured under nitrogen. Therefore $\Delta R_{V_S} = \frac{R_{V_S}}{R_0}$ is calculated by :

- First derive the resistance under nitrogen R_0
- Second devide all resitance values by this value

[48]:

```python
from scipy.optimize import curve_fit
from scipy.interpolate import interp1d



fig, axe = subplots(1, figsize=(16,10))

#get the value of the flatband (if needed)
#by interpolation
interp_res = interp1d(v_exp,res_exp)
res_flatband = interp_res(0)

#calcualte the rel. res change
rel_res_exp = dF['res']/res_flatband



#represent it
```

```
format_axe(axe)
axe.scatter(dF['dV'],rel_res_exp, s=100)
axe.set_ylim(rel_res_exp.min()/2,
             rel_res_exp.max()*2);
```



2.4 Interpolating the numerical values

In the previous section, the numerical solution for multiple start parameters have been calculated. Nevertheless most probably the calculated dataset will not hold exactly the same values gathered from the experiment. To obtain the numerical value for a specific experimental value of qV_s a interpolation between of the existing numerical values will be used again. For all different numerical grains ΔR_{V_S} will be calculated for all the experimental values of qV_s. Once this is done, the different between the numerical model and the exp. data can be calculated and evaluated.

[49]:
```
#The dataframe to hold the different
#of the exp. values to the numerical ones
#Will be used to find the best fitting num. solution
num_data_at_exp_pos_dF = pd.DataFrame(index = v_exp)

#group the num. data by its paramters (T, R and ND)
data_by_grain = calc_dF.groupby(['temp','R','ND'])

for (T, R,ND), calc_dF_grain in data_by_grain:

    num_data_at_exp_pos_dF[(T, R,ND)] = None

    grain = create_grain_from_data(calc_dF_grain)

    flat_band_data = calc_dF_grain[calc_dF_grain['Einit_kT']==0].iloc[0]

    rel_res_num = calc_dF_grain['rel_res_change']

    #express the surace potential in eV
    #to be comparable with the exp. data
    v_num = calc_dF_grain['Einit_kT']*CONST.J_to_eV(grain.material.kT)

    #use interpolation to get the values for the positions
    #of the experiment data points
    interp_rs_num = interp1d(v_num, rel_res_num,bounds_error=False)
    interp_v_num = interp1d(rel_res_num,v_num, bounds_error=False)

    #caculate the numerical value of rel. res at the position
    # of V from the experiment
    res_num_at_exp_pos = interp_rs_num(v_exp)

    #save those values in the new DataFrame
    num_data_at_exp_pos_dF.loc[:,(T, R,ND)] = res_num_at_exp_pos
```

2.5 Calculating the fit error

num_data_at_exp_pos_dF contains now the values of ΔR_{V_S} at the positions qV_S. From these values the relative error needs to be calculated. The following formula is used to derive the error:

$$\epsilon_{V_S} = \left(\frac{R_{numerical}(qV_s) - R_{experiment}(qV_s)}{R_{experiment}(qV_s)} \right)^2 \tag{1}$$

The sum of all ϵ_{V_S} is the total error of the fit. The numerical model with the lowest value of $\sum \epsilon_{V_S}$ is the model which fits best to the experimental data. The average grain diameter of the material “IPC100” is known to be in average 110nm (Nanoparticle engineering for gas sensor optimization: Improved sol-gel fabricated nanocrystalline SnO2 thick film gas sensor for NO2 detection by calcination, catalytic metal introduction and grinding treatments, 1999). The dataset we created in the previous section includes models for grains with radii of 50nm and 100nm. Therefore we can narrow the fit algorithm down, to take only models with a radius of 50nm and 100nm in account

[50]:

```
abs_error = num_data_at_exp_pos_dF.subtract(rel_res_exp, axis='index')
rel_error = abs_error.divide(rel_res_exp, axis='index')
rel_error_square = rel_error**2
sum_of_squares = rel_error_square.sum()

valid_index = [i for i in sum_of_squares.index if i[1] in [50e-9,100e-9]]
```

```
sum_of_squares_grainsize = sum_of_squares.loc[valid_index].sort_values()

grain_min_error_tuple = sum_of_squares_grainsize.idxmin()
display(pd.DataFrame({'error':sum_of_squares_grainsize}))
grain_min_error_tuple
```

	error
(300.0, 5e-08, 1e+22)	6.82
(300.0, 1e-07, 1e+22)	7.67
(300.0, 1e-07, 1e+21)	14.36
(300.0, 5e-08, 1e+21)	18.77
(300.0, 5e-08, 1e+23)	175.20
(300.0, 1e-07, 1e+23)	1256.74
(300.0, 5e-08, 1e+24)	9905.88
(300.0, 1e-07, 1e+24)	39709.78

```
[50]: (300.0, 5e-08, 1e+22)
```

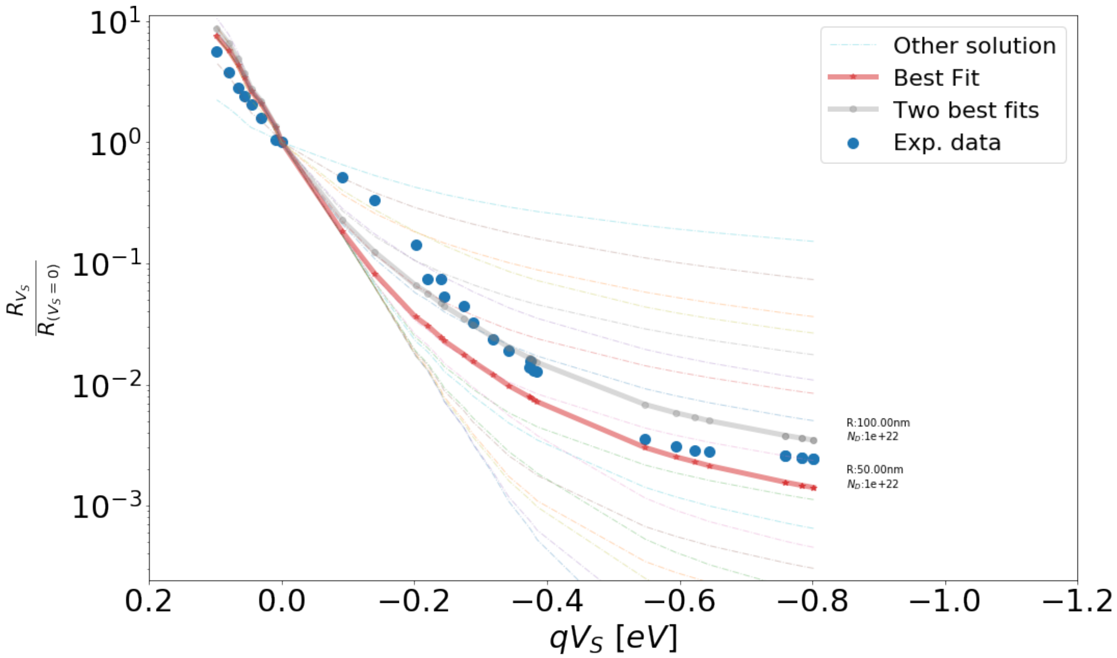
2.6 Representation of the fit

Finally the best fit results can be represented graphically.

[54]:

```
fig, axe = subplots(figsize = (16,10))
#for grain_tuple in num_data_at_exp_pos_dF.keys():
for grain_tuple in sum_of_squares.index:
    if grain_tuple == grain_min_error_tuple:
        linestyle = '*-'
        linewidth = 5
        alpha = 0.5
        label = 'Best Fit'
    elif grain_tuple in sum_of_squares_grainsize.index[0:2]:
        linestyle = '-o'
        linewidth = 5
        alpha = 0.3
        label = 'Two best fits'
    else:
        linestyle = '-.'
        linewidth = 1
        alpha = 0.3
        label = 'Other solution'




    axe.plot(num_data_at_exp_pos_dF.index,
                num_data_at_exp_pos_dF[grain_tuple],
            linestyle, linewidth=linewidth, alpha = alpha,
            label =label)


    last_x  = num_data_at_exp_pos_dF.index[0]
    last_y  = num_data_at_exp_pos_dF.iloc[0][grain_tuple]

    if grain_tuple in sum_of_squares_grainsize.index[0:2]:
        axe.text(last_x-0.05,last_y,
            f'''R:{grain_tuple[1]*1e9:.2f}nm\n$N_D$:{grain_tuple[2]:.2}''')
format_axe(axe)

axe.scatter(rel_res_exp.index,
            rel_res_exp,
            s=100,
            label = 'Exp. data'
           )

axe.set_ylim(rel_res_exp.min()/10,
             rel_res_exp.max()*2);


l = {h[1]:h[0] for h in zip(*axe.get_legend_handles_labels())}.keys()
h = {h[1]:h[0] for h in zip(*axe.get_legend_handles_labels())}.values()
axe.legend(h,l,loc=1, fontsize = 22)
```

```
[54]: <matplotlib.legend.Legend at 0x7fb19c3de490>
```



3 Conclusion

We can see, that both fits do not fit perfectly to the experimental data. One obvious reason is the curse screening of the grain size and and N_D. A second iteration of creating models with radii between 50nm and 100nm might find a better fit. Also a finer screening of N_D will turn out to be helpful for a better result. On the other side the fitting shows, that the experimental data fits will to a grain with approximately 50nm radius and a defect concentration of around $N_D = 1 * 10^{22} 1/m$ at 300°C.

4 Bibliography section

References

[Reb16] REBHOLZ, Julia M.: *Influence of Conduction Mechanism Changes and Related Effects on the Sensing Performance of Metal Oxide Based Gas Sensors*. Shaker Verlag, 2016. – 127 S. – ISBN 978–3–8440–4832–2