CMP_SC_RUN_EXAMPLE

September 19, 2022

1 Superconductivity - Condesed Matter Physics

1.0.1 PHY424 - University of Toronto Advanced Physics Labs

1.0.2 Program How To

This tutuorial walks you through how to use to programs to take a data analysis run on the CMP superconductivity experiment.

The power in this program is the ability to change the sample rate of the data aquisition whih will effect the resolution of the I-V curve near the superconducting transition temperature.

[1]:
```
# import these libraries
import numpy as np
from event_handler import EventHandler
import time
import sys
```

[2]:
```
# Create an event object
cmp_sc_run = EventHandler()
```

Run the system up to a transdiode reading of 0.8 volts at a sample rate of 10 ms.

[5]:
```
cmp_sc_run.run(0.8, 10000)
```

```
Begining data aquisition…
Lock in running.
Run complete.
```

The lock in data is stored in a dictionary, whose key value is a number corresponding to the data type and the value is a list of stored measurements.

key	measuremnt type
0	X magnitude
1	Y magnitude
3	Phase
4	Sensitivty
5	Noise

The scope data is stored in the a list. By default this is a list of voltage readings from the transdiode.

[6]:
```
lock_in_data = cmp_sc_run.lock_in.data
scope_data = cmp_sc_run.scope.data
```

[6]:
```
scope = cmp_sc_run.scope
```

[8]:
```
 mag = lock_in_data[0]  # x magnitude
```

[10]:
```
import matplotlib.pyplot as plt
```

[20]:
```
SEN = {1: 2e-9, 2: 5e-9, 3: 10e-9,
       4: 20e-9, 5: 50e-9, 6: 100e-9,
       7: 200e-9, 8: 500e-9, 9: 1e-6,
       10: 2e-6, 11: 5e-6, 12: 10e-6,
       13: 20e-6, 14: 50e-6, 15: 100e-6,
       16: 200e-6, 17: 500e-6, 18:1e-3,
       19: 2e-3, 20: 5e-3, 21: 10e-3,
       22: 20e-3, 23: 50e-3, 24: 100e-3,
       25: 200e-3, 26: 500e-3, 27: 1}
```

[21]:
```
# convert sensitivity output from lock in to decimals
sen = [SEN[i] for i in lock_in_data[4]]
```

[23]:
```
# convert x reading and sensitivity (float converted) to arrays
# and complete element wie multiplications
M = np.array(mag)
S = np.array(sen)
X = M * S
```

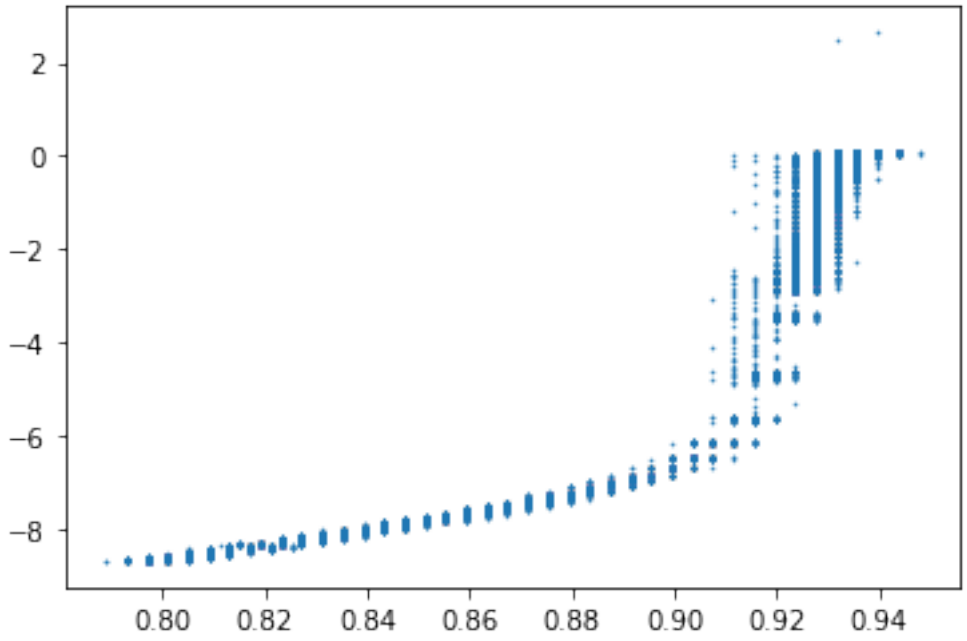
Simply plotting the results may give a weird looking graph. Note the below the temperatre of the transdiode attached to the sample is in voltages with 0.99 representing 77 kelvin. 1) The first issue is that when we convert the sensitivy to a float for some reason we must divide the values by another factor of 10. I am not sure why but this does give correct results.

2) In this case the sign of the voltage is negative, but rememeber we do not care about the sign only the absolute value.

3) There is also some aliasing coming in that we would like to average over.

[34]:
```
plt.scatter(scope_data, np.abs(X), s=0.3)
```

[34]:
```
<matplotlib.collections.PathCollection at 0x11fe17be0>
```



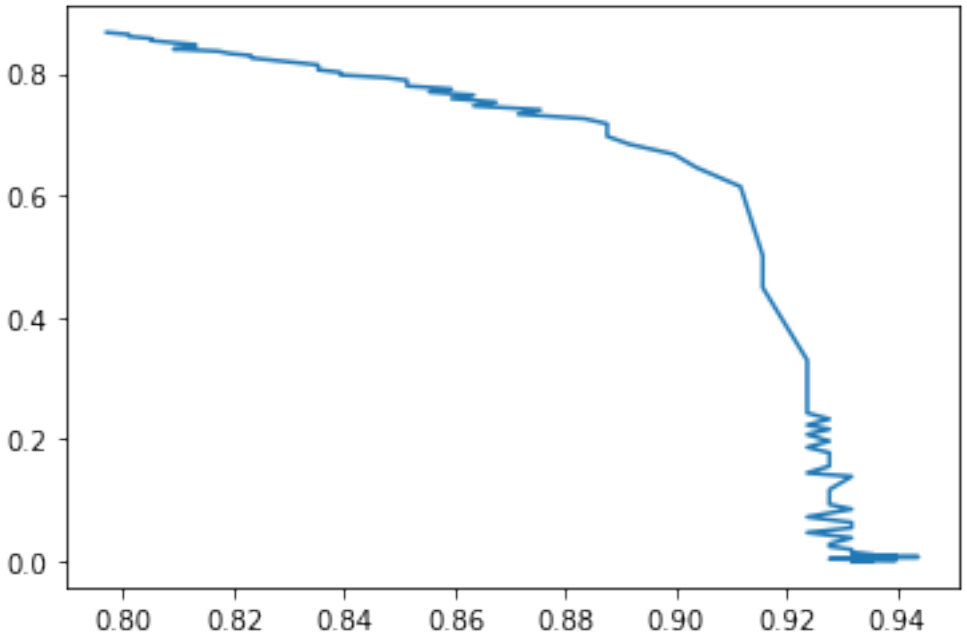
Now we average over the each iteration (length of buffer is 300) and look at the absolute value of the averaged x magnitude. We see that the trend matches what we would expect. As the material decreases in temperature the resistiviy of the material decreases (ie the voltage).

Remember we are looking at Vrms here to get a reading of the resistance (or more generally impedeance) because the Irms is known. To actually get the resistance value (meausrements in ohms) you have to divide by the current rms.

[35]:
```
k = []
for i in range(0, 30900, 300):
    k.append(np.average(X[i:i+301]))
```

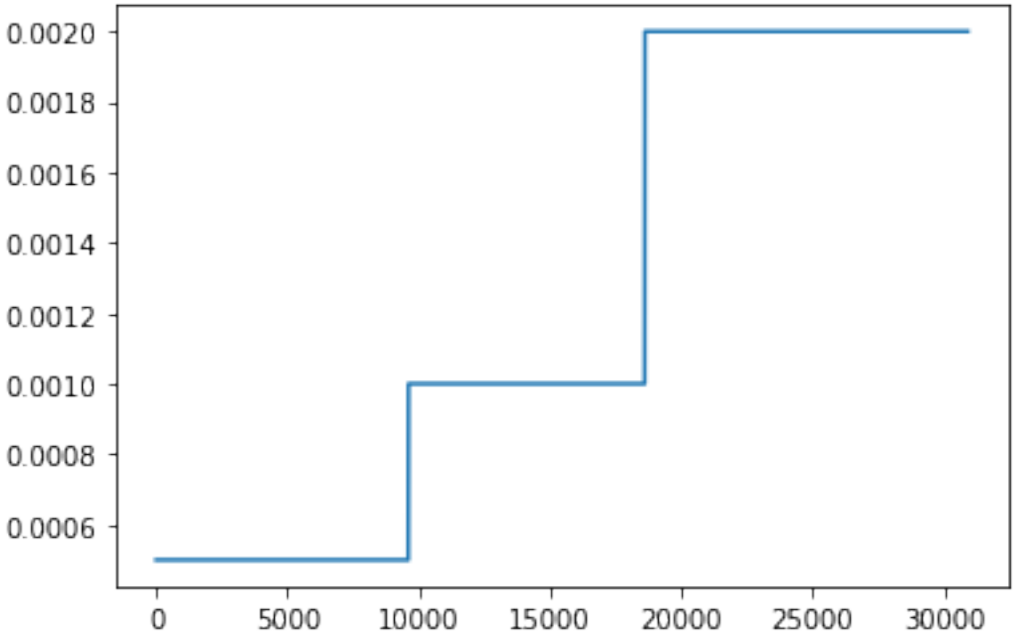
[42]:
```
plt.plot(scope_data[::300], np.abs(k)*1e-1)
```

[42]:
```
[<matplotlib.lines.Line2D at 0x120398850>]
```



[43]:
```
# here is a plot of the Sensitivity
plt.plot(S)
```

[43]: [<matplotlib.lines.Line2D at 0x1203f1ee0>]



[44]:
```
# now we save the data
np.savetxt("lock_in_data_x.csv", lock_in_data[0])
```

[45]:
```
np.savetxt("lock_in_data_y.csv", lock_in_data[1])
np.savetxt("lock_in_data_phase.csv", lock_in_data[3])
np.savetxt("lock_in_data_sen.csv", lock_in_data[4])
np.savetxt("lock_in_data_noise.csv", lock_in_data[5])
```

[46]:
```
np.savetxt("scope_data.csv", scope_data)
```

[ ]:
```

```