For_Git_FM112_Flake_AHE

Imports

[13]:
```
import numpy as np
from numpy import loadtxt
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import glob
from numpy import diff
import pandas as pn
import math
import scipy.constants as sc
import pickle
import copy
from scipy import interpolate
from matplotlib import rcParams, cycler, cm, rc
plotall = True
overview_plot = True
from pylab import␣
↪meshgrid,cm,imshow,contour,clabel,colorbar,axis,title,show,pcolor
import pandas as pd
from matplotlib.lines import Line2D
import scipy as scipy
```

Selects the appropiate data and gives it a file path. Also prints the path and the amount of files in its folder

[15]:
```
# --- Selecting data from AHE Folder ---
folder_AHE = r"C:\Users\pblah\Data\Navy Beach\FM112 Flake\AHE"
subpath_AHE = folder_AHE + "\*.txt"
print(subpath_AHE)
pathlist_AHE = glob.glob(subpath_AHE)
F = int(len(folder_AHE)) ##this is used for calculating the number of characters␣
↪in folder name to find T
print(len(pathlist_AHE))

# ---Selecting data from MR folder ---
folder_MR = r"C:\Users\pblah\Data\Navy Beach\FM112 Flake\MR"
subpath_MR = folder_MR + "\*.txt"
```

```
print(subpath_MR)
pathlist_MR = glob.glob(subpath_MR)
F = int(len(folder_MR)) ##this is used for calculating the number of characters␣
↪in folder name to find T
print(len(pathlist_MR))

print(pathlist_MR)
```

```
C:\Users\pblah\Data\Navy Beach\FM112 Flake\AHE\*.txt
0
C:\Users\pblah\Data\Navy Beach\FM112 Flake\MR\*.txt
0
[]
```

Finds the Temperatures used via the filename and makes them into an array

[17]:
```
# --- Finding Temperature Array for data in certain folder ---

Temperature_list = []
for i,path in enumerate(pathlist_AHE[0::]):
    file = path[F::]
    T_index_max = file.find('K.')
    string_tmp = file[T_index_max-5:T_index_max]
    T_index_min = string_tmp.find('_')
    Temperature = string_tmp[T_index_min+1::]
    Temperature=float(Temperature)
    Temperature_list = np.append(Temperature_list,Temperature)
print(Temperature_list)

print(type(Temperature_list))
```

```
[  1.5   20.1   39.9   59.9   79.8   89.8   99.9 109.9 120.   124.8 127.   130.
 140.   159.9]
<class 'numpy.ndarray'>
```

0.0.1 Closest Element Function

[18]:
```
def closest_element(array,value):
    element = min(array, key=lambda x:abs(x-value))
    closest_element = np.where(array == element)[0][0]
    return closest_element
```

0.0.2 Closest Element Index Function

[19]:
```
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
```

1 Anti-Symmetriser

1.1 Anomalous Hall Resitivity

Extracts B and Resistance. Converts to resitivity (formula depends on VdP or Hall). Splits data into 4 branches, maps it to a common x-axis. Anti-Symmetrises them.

[13]:
```
#cm = plt.get_cmap('winter', 18) # colour map

#Temperature labels for legend
Temperature_labels = Temperature_list.astype(str)

#Setting Size of figure 1
fig, ax = plt.subplots(figsize=(36, 12), dpi = 500)



offset = 0

for i,path in enumerate(pathlist_AHE[0::]): #Looping over AHE data from folder

    data = np.loadtxt(path, skiprows=2) #Extracting data from txt file in form␣
↪of list

    B_raw = data[:,1]
    R_xx_raw = data[:,2]
    Theta_xx_raw = data[:,3]
    R_xy_raw = data[:,4]
    Theta_xy_raw = data[:,5]

    Rxx_raw = R_xx_raw*np.cos(Theta_xx_raw*sc.pi/180)  # Working out the␣
↪resistances taking the lock-in phase into account
    Rxy_raw = R_xy_raw*np.cos(Theta_xy_raw*sc.pi/180)


    B_max = np.max(B_raw)  #The maximum B field in the dataset

    Delta_B = 0.005 # in Tesla
    pts = int(B_max / Delta_B+1) #These points are used later to create my␣
↪simulated x-axis(aka.B Field)
```

```python
    index_max = np.where(B_raw>=B_max)[0] #The datapoints where the B field is at a maximum, so a few at the BEGINNING and a few at the END of the sweep
    index_min = np.where(B_raw<=-B_max)[0] #The datapoints where the B field is at a minimum, so a few at the MIDDLE of the sweep

    MAX = index_max[int(len(index_max))-1] #Takes the last one of these points that are at the maximum. The -1 is there because the slice function [A:B] starts from the point AFTER A and ends at the point one BEFORE B
    MIN = index_min[0] #Just picks the first value of these few points that are at the mimumum



    #---Not sure this if/else condition is needed now, if I do the sweeps as +10 to -10 to +10, then my last index max point will always be larger than index min?---

    if index_min[0] > index_max[0]: # This if/else condition determines what to do with the data depending on whether you're looking at the data from the BEGINNING of the sweep to halfway(+10 to -10), or halfway to the END of the sweep (-10 to +10)
        B_new = np.linspace(-B_max, B_max,pts)
        B = B_raw[MIN:MAX]
        R_xx = R_xx_raw[MIN:MAX]
        Theta_xx = Theta_xx_raw[MIN:MAX]
        R_xy = R_xy_raw[MIN:MAX]
        Theta_xy = Theta_xy_raw[MIN:MAX]

    else:
        B_new = np.linspace(B_max, -B_max,pts)
        B = B_raw[MAX:MIN]
        R_xx = R_xx_raw[MAX:MIN]
        Theta_xx = Theta_xx_raw[MAX:MIN]
        R_xy = R_xy_raw[MAX:MIN]
        Theta_xy = Theta_xy_raw[MAX:MIN]


    Rxx = R_xx*np.cos(Theta_xx * np.pi /180) #Works out the resistances over the range MAX to MIN or MIN to MAX
    Rxy = R_xy*np.cos(Theta_xy * np.pi /180)

    Rxy_raw = Rxy_raw *1E6* 2E-6/1.5E-6 * 5.5E-9*1E2 # Anomalous Hall resitivity
```

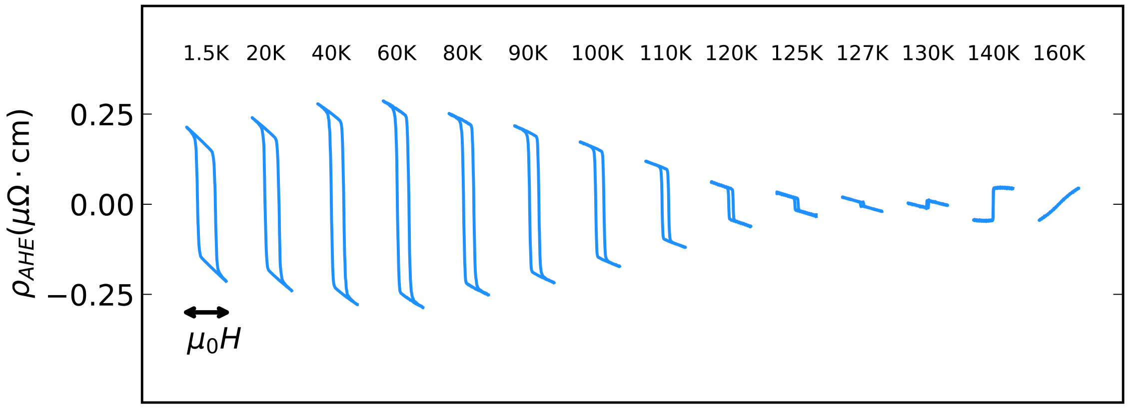
```
IZero=np.where(B_raw==0)[0] #The index list of values you get at B=0. The [0] is there because without it it spits out an array that contains a list, when we just want the list(ie. the first element of that array)

Imax=np.where(B_raw==np.max(B_raw))[0][0] #The (index of) the first value of the maximum of the B field(+10 here), ie. the beginning of the sweep, so [0][0] is the first element of that list which is the first element of the array

Imin=np.where(B_raw==np.min(B_raw))[0][0] #The (index of) the first point of where the B field is at a minumum, ie. the middle of the sweep

FZ=IZero[0] #The datapoint when you go through B=0 for the first time
SZ=IZero[-1] #The datapoint when you go through B=0 for the second time


Rxy_1 = Rxy_raw[Imax:FZ+1] #The first branch of the sweep. You do +1 due to the slice notation not using inclusive values (explained in previous cell)
B_1 = B_raw[Imax:FZ+1] #The x-axis of this first branch

Rxy_2 = Rxy_raw[FZ:Imin+1] #The second branch of the sweep
B_2 = B_raw[FZ:Imin+1] #The x-axis of this second branch

Rxy_3 = Rxy_raw[Imin:SZ] #The third branch of the sweep
B_3 = B_raw[Imin:SZ] #The x-axis of this third branch

Rxy_4 = Rxy_raw[SZ::] #The fourth branch of the sweep
B_4 = B_raw[SZ::] #The x-axis of this fourth branch

TM=np.where(B_raw[Imax:FZ+1]==np.max(B_raw[Imax:FZ+1]))[0] #At the beginning, (note also at the end), there are a few trailing values. Ie. the sweep will go 10,10,10,9.9,9.8 etc. This gives the index list of these values
B_int = B_raw[TM[-1]:FZ+1] #This gives the positive x-axis of the sweep (10 to 0) while only picking one of the trailing values. Note only the positive side is picked as when you sym/antisym the full data, it'll end up being only on one side



f_1 = interpolate.interp1d(B_1, Rxy_1) #This creates a linear interpolation (aka. a line between nearest neighbour points) of the data in the first branch
Rxy_1_int = f_1(B_int) #This maps this line to the x-axis which will be common to all four branches

f_3 = interpolate.interp1d(B_3, Rxy_3) #This creates a linear interpolation (aka. a line between nearest neighbour points) of the data in the third branch
```

```
    Rxy_3_int = f_3(-B_int[::-1]) #This begins the mapping of this line to the␣
↪x-axis which will be common to all four branches. The -B_int[::-1] means; so␣
↪-B_int is -5, -4.9...-0. Then [::-1] means you reverse the order so -0, -0.1...
↪-5.
    Rxy_3_int = Rxy_3_int[::-1] #You then reverse the order again, so you go␣
↪from -5,-4.9...-0. This is because the last point of this branch needs to␣
↪correspond to the first point of the fourth branch.

    f_2 = interpolate.interp1d(B_2, Rxy_2) #This creates a linear interpolation␣
↪(aka. a line between nearest neighbour points) of the data in the second␣
↪branch
    Rxy_2_int = f_2(-B_int[::-1]) #This begins the mapping of this line to the␣
↪x-axis which will be common to all four branches. The -B_int[::-1] means; so␣
↪-B_int is -5, -4.9...-0. Then [::-1] means you reverse the order so -0, -0.1...
↪-5.
    Rxy_2_int = Rxy_2_int[::-1] #You then reverse the order again, so you go␣
↪from -5,-4.9...-0. This is because the last point of this branch needs to␣
↪correspond to the first point of the fourth branch.

    f_4 = interpolate.interp1d(B_4, Rxy_4) #This creates a linear interpolation␣
↪(aka. a line between nearest neighbour points) of the data in the fourth␣
↪branch
    Rxy_4_int = f_4(B_int) #This maps this line to the x-axis which will be␣
↪common to all four branches



    #Anti-Symmetrising the data using the four interpolated branches

    Asym_NH_pos = (Rxy_1_int - Rxy_3_int)/2 #NH means Normal Hall, pos means the␣
↪positive side of the x axis. This gives the difference between the branches␣
↪which are mostly due to the normal Hall effect
    Asym_AHE_pos = (Rxy_4_int - Rxy_2_int)/2 #This gives the difference between␣
↪the branches which are mostly due to the Anomalous Hall effect

    Asym_NH_neg = -Asym_NH_pos #Because anti-symmetrising the data converts the␣
↪4 branches into two, I've done it so that they map to the positive x axis.␣
↪Thus for display reasons I am taking the negative of these branches to display␣
↪the 'full curve'
    Asym_AHE_neg = -Asym_AHE_pos

    #---Combining the 4 antisymmetrised branches into one full curve---

    full_curve = np.append(Asym_NH_pos,Asym_AHE_neg[::-1])
    full_curve1 = np.append(full_curve,Asym_NH_neg)
    full_curve2 = np.append(full_curve1,Asym_AHE_pos[::-1])
    full_B = np.append(B_int,-B_int[::-1])
```

```
    full_B1 = np.append(full_B,-B_int)
    full_B2 = np.append(full_B1,B_int[::-1])


    if i ==0 or i==1 or i==2 or i==3:

        minus2T_range = closest_element_index(full_B2, -1.5)
        plus2T_range = closest_element_index(full_B2, 1.5)


        #print(minus2T_range)
        #print(plus2T_range)

        minus2T_1 = int(minus2T_range[0])
        minus2T_2 = int(minus2T_range[2])
        plus2T_1 = int(plus2T_range[0])
        plus2T_2 = int(plus2T_range[2])

        #print(plus2T_1,minus2T_1)
        #print(plus2T_2,minus2T_2)



    #Plotting it with colour convention and saving the data to a pandas Dataframe


        #print(full_B2[plus2T_1:minus2T_1])
        #plt.plot(full_B2[plus2T_1:minus2T_1] + offset, full_curve2[plus2T_1:
↪minus2T_1], ms=1,alpha=1,color = 'dodgerblue', lw=4)
        #plt.plot(full_B2[minus2T_2:plus2T_2] + offset, full_curve2[minus2T_2:
↪plus2T_2], ms=1,alpha=1,color = 'dodgerblue', lw=4)#color=cm(i/
↪len(Temperature_labels))

        full_B_df = np.concatenate((full_B2[plus2T_1:
↪minus2T_1],full_B2[minus2T_2:plus2T_2]))
        full_curve_df = np.concatenate((full_curve2[plus2T_1:
↪minus2T_1],full_curve2[minus2T_2:plus2T_2]))

        #plt.plot(full_B_df + offset, full_curve_df, ms=1,alpha=1,color =␣
↪'dodgerblue', lw=4)
        plt.plot(full_B_df + offset, full_curve_df, ms=1,alpha=1,color =␣
↪'dodgerblue', lw=6)

        pd.DataFrame({'B':full_B_df,'AHE':full_curve_df}).to_csv(r'C:
↪\Users\pblah\Data\Navy Beach\Data for Combined Plots\AHE\FM112 Flakes\FM112␣
↪Flakes.csv')
```

```
        pd.DataFrame({'B':full_B_df + offset,'AHE':full_curve_df}).to_csv(r'C:
↪\Users\pblah\Data\Navy Beach\Data for Combined Plots\Giga AHE Plot\FM112␣
↪Flakes\FM112 Flakes ' + str(i) + '.csv')


    else:
        #plt.plot(full_B2+ offset, full_curve2, ms=1,alpha=1,color =␣
↪'dodgerblue', lw=4)
        plt.plot(full_B2+ offset, full_curve2, ms=1,alpha=1,color =␣
↪'dodgerblue', lw=6)
        pd.DataFrame({'B':full_B_df + offset,'AHE':full_curve_df}).to_csv(r'C:
↪\Users\pblah\Data\Navy Beach\Data for Combined Plots\Giga AHE Plot\FM112␣
↪Flakes\FM112 Flakes ' + str(i) + '.csv')
        #pd.DataFrame({'Temperature':Temperature_labels1,'Coercive_Field':
↪coercive_field_append}).to_csv(r'C:\Users\pblah\Data\Navy Beach\Data for␣
↪Combined Plots\Coercive Field\FM112 Flake.csv')


    offset = offset + 5 #This offset is to allow all the loops to be plotted on␣
↪one graph

    rho_xy_atzeroB = full_curve2[IZero][0] #Value of AHE where B=0

    #print('rho_xy_atzeroB',rho_xy_atzeroB)





###################  Annotations  ####################################

    ###  B Arrow Annotations  ###
plt.annotate("",xy=(-2, -0.3), xycoords='data', xytext=(2, -0.3),␣
↪textcoords='data',  arrowprops=dict(arrowstyle = '<->', connectionstyle =␣
↪"arc3", linewidth = 9, mutation_scale = 40),)
plt.annotate("$\mu_0 H$",xy = (-1.5,-0.4),fontsize=60, weight = 'bold')

    ###  Temperature annotations ###
plt.annotate(str(Temperature_list[0]) + "K",xy = (-1.8,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[1])) + "K",xy = (3,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[2])) + "K",xy = (8,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[3])) + "K",xy = (13,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[4])) + "K",xy = (18,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[5])) + "K",xy = (23,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[6])) + "K",xy = (27.8,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[7])) + "K",xy = (33,0.4),fontsize=42)
```

```python
plt.annotate(str(round(Temperature_list[8])) + "K",xy = (38,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[9])) + "K",xy = (43,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[10])) + "K",xy = (48,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[11])) + "K",xy = (53,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[12])) + "K",xy = (58,0.4),fontsize=42)
plt.annotate(str(round(Temperature_list[13])) + "K",xy = (63,0.4),fontsize=42)

# Borders#

ax.spines["top"].set_linewidth(5)
ax.spines["bottom"].set_linewidth(5)
ax.spines["right"].set_linewidth(5)
ax.spines["left"].set_linewidth(5)
ax.spines["left"].set_bounds(-0.55,0.55)
ax.spines["right"].set_bounds(-0.55,0.55)
ax.spines["top"].set_position(['data',0.55])
ax.spines["bottom"].set_position(['data',-0.55])


#Axis Limits#

plt.ylim(-0.45,0.45)


########  Labels, ticks and saving  #############

#plt.title("FM112 Flake AHE",fontsize=30)
#ax.legend(Temperature_list,ncol=2,fontsize=15,loc =1)
#plt.legend(Temperature_labels)
#plt.xlabel(r'$\mu _0 H_\perp$(T)',fontsize=30)
plt.ylabel(r'$\rho_{AHE}(\mu\Omega\cdot$cm)',fontsize=60, labelpad = 20)
ax.tick_params(axis='y', which='major', labelsize=80, length = 20, width = 2,␣
↪direction = 'in', pad = 15, right = True)
plt.xticks([])
plt.yticks(fontsize = 60)
#plt.savefig(r"C:\Users\pblah\Data\Navy Beach\FM112␣
↪Flake\Figures\FM112_Flake_AHE_Curves.pdf",bbox_inches = "tight")
#plt.savefig(r"C:\Users\pblah\Data\Navy Beach\FM112␣
↪Flake\Figures\FM112_Flake_AHE_Curves.png",bbox_inches = "tight")
plt.savefig(r"C:\Users\pblah\Data\Navy Beach\FM112␣
↪Flake\Figures\FM112_Flake_AHE_Curves_PRB.pdf",bbox_inches = "tight", format =␣
↪"pdf")
plt.savefig(r"C:\Users\pblah\Data\Navy Beach\FM112␣
↪Flake\Figures\FM112_Flake_AHE_Curves_PRB.png",bbox_inches = "tight")
plt.show()
```


1.5K
20K
40K
60K
80K
90K
100K
110K
120K
125K
127K
130K
140K
160K
0.25
0.00
−0.25
$\rho_{AHE}(\mu\Omega\cdot\mathrm{cm})$
$\mu_0 H$