Hysteresis_analysis_Gregor

February 26, 2022

0.1 Hysteresis lab - Benedikt Gregor

Physical measurements of components were taken with callipers Iron core measurements width, callipers +-0.01mm: 29.98, 30.18, 30.00, ruler: 31mm +-0.5mm height iron core, callipers +-0.01mm: 29.00, 29.00, 29.00, ruler: 28mm +-0.5mm length iron core, callipers +-0.01mm: 101.62, 101.50, 101.52, ruler : 101mm +-0.5mm

steel core length, callipers +-0.01mm: 102.82, 102.92, 102.90, ruler: 110mm +-0.5mm

width steel core, callipers +-0.01mm: 28.00, 27.98, 28.00, ruler: 27mm +-0.5mm

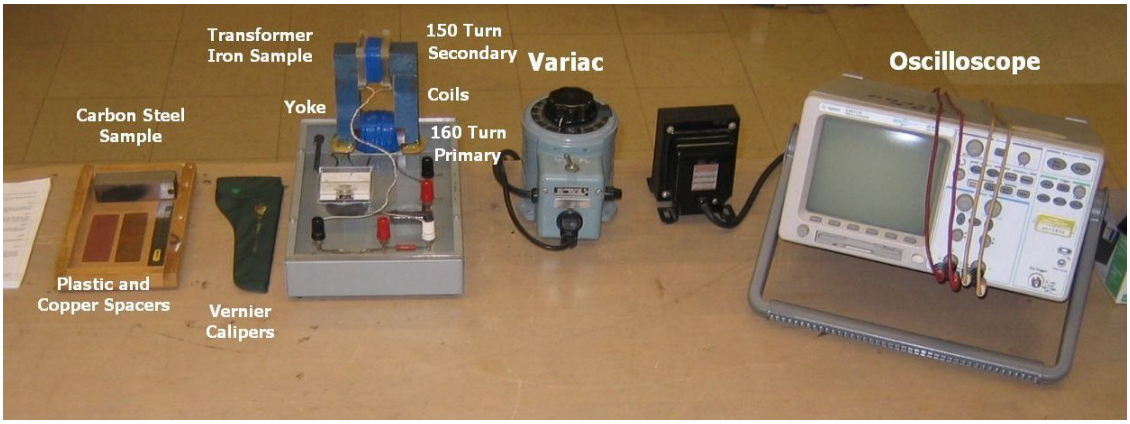
steel core height, callipers +-0.01mm: 27.56, 27.62, 27.50, ruler: 27mm +-0.5mm

height of yoke, callipers: 150.00, 150.00, 150.00

length of yoke, callipers +-0.01mm: 101.70, 101.90, 101.98

width of yoke, callipers: 30.40, 30.22, 30.10

Setup:



After the measurements we turned on the oscilliscope (2534 digital storage oscilloscope bk precision) and set channel 2 to a function of channel one, pressing horizontal and select x-y. Then we used a BNC to banana plug adapter and inserted it into channel one. Then the cables for the H field coming from the transparent box were plugged into the adapter for the x axis. Then was the B field plugged in for channel 2 also with an adapter and two banana cables.

We slid the iron sample into the secondary coil so that it is centered and then placed on the top of the yoke. With the variac set to 0 we turned it on and gradually increased the voltage. With the increase in voltage the oscillations in the yoke caused an ever louder buzzing sound. The hysteresis

loop started to appear gradually from the center of the oscilloscope screen. It spread from the middle into its shape. To make the curve smoother the averaging function on the oscilloscope was used by pressing acquire and average for 256 cycles. The voltage was then further increased until the curve spans 400mV on the x axis. The scale was between 50 and 60 on the variac and the amperemeter read 3 on the A.C. scale. Next, the oscilloscope was set back to the time domain by selecting horizontal, main. The time resolution was set to 5ms per division. Then a USB stick was plugged into the oscilloscope and data was saved with save/load, external storage, new, file type csv.

Iron sample loop progression: The oscilloscope was returned to x-y mode again to examine the hysteresis loop progression. The Variac was turned and voltage was increased to about 30 on the scale with an aperemeter reading about 0.5. Using the manual cursor with channel one as the source and the scale set to CH1=50mV and CH2=100mV data points were collected. The variac is then increased to 40 on the scale with the amperemeter reading about 1 and the scale is changed to CH1=100mV. Followed up with an increase to 50 with the amperemeter showing about 1.5. Between those settings at least 15 data points have to be recorded.

Iron sample energy losses: The temperature of the iron sample was measured with an infrared temperature gun by mastercraft reading about 41.2°C. For comparison a 40W lightbuld was also measured which resulted in 42.5 °C.

Next the effects of plastic and copper spacers on the loop were tested. The spacer was slid between the iron core and the yoke so that it also passes through the secondary coil. With the plastic inserted the variac was turned up to around 70 (5.4 on amp scale) and the hysteresis loop looked more like a curve now with no descernable area using scales: CH1=200mV and CH2=200mV The copper spacer was only turned up on the variac to about 15 (1.8 on amp) on the scale. The instruments started shaking under the vibrations and the hysteresis loop had become oval shaped, or in other words a big increase in area. This was observed with the scales: CH1=100mV and CH2= 20mV

The same steps are now conducted with the steel sample. Turning the variac up to about 50 again (3 on amp) the hysteresis curve immediatly looks like it has more area than the iron one with scale CH1=100mV and CH2=100mV. When not turning off the variac without reducing the voltage to zero beforehand the sample becomes hard to remove as it magnetically still sticks to the yoke. Judging from the resolution of the oscilloscope and the adjustments of the cursor the error should be about +-2mV.

[138]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 150
import pandas as pd

# iron sample measurements
iron_width = np.array([29.98, 30.18, 30.00])
iron_height = np.array([29.00, 29.00, 29.00])
iron_length = np.array([101.62, 101.50, 101.52])

steel_width = np.array([28.00, 27.98, 28.00])
steel_height = np.array([27.56, 27.62, 27.50])
steel_length = np.array([102.82, 102.92, 102.90])
```

```
yoke_width = np.array([30.40, 30.22, 30.10])
yoke_height = np.array([150.00, 150.00, 150.00])
yoke_length = np.array([101.70, 101.90, 101.98])

data1 = np.column_stack((iron_width, iron_height, iron_length))
data2 = np.column_stack((steel_width, steel_height, steel_length))
data3 = np.column_stack((yoke_width, yoke_height, yoke_length))
#print(data1)

# getting averages and converting mm to m
iron_w, iron_h, iron_l = np.average(iron_width)/1000, np.average(iron_height)/
 ↪1000, np.average(iron_length)/1000
steel_w, steel_h, steel_l = np.average(steel_width)/1000, np.
 ↪average(steel_height)/1000, np.average(steel_length)/1000
yoke_w, yoke_h, yoke_l = np.average(yoke_width)/1000, np.average(yoke_height)/
 ↪1000, np.average(yoke_length)/1000
# error
iron_w_error, iron_h_error, iron_l_error = np.std(iron_width)/1000, np.
 ↪std(iron_height)/1000, np.std(iron_length)/1000
steel_w_error, steel_h_error, steel_l_error = np.std(steel_width)/1000, np.
 ↪std(steel_height)/1000, np.std(steel_length)/1000
yoke_w_error, yoke_h_error, yoke_l_error = np.std(yoke_width)/1000, np.
 ↪std(yoke_height)/1000, np.std(yoke_length)/1000

Aci = iron_w*iron_h # corss-sectional area for iron sample
Acs = steel_w*steel_h # corss-sectional area for steel sample

Aci_error = np.sqrt((iron_w_error/iron_w)**2 + (iron_h_error/iron_h)**2)*Aci
Acs_error = np.sqrt((steel_w_error/steel_w)**2+(steel_h_error/steel_h)**2)*Acs

# making a table to nicely read raw data
def table(data1, N):
    fig, ax = plt.subplots(1,1)
    column_labels = ["width [mm] +-0.01", "height [mm] +-0.01", "length [mm] +-␣
 ↪0.01"]
    df =pd.DataFrame(data1, columns=column_labels)
    ax.axis('tight')
    ax.axis('off')
    if N == "I":
        plt.title("Iron sample dimensions")
    elif N == "S":
        plt.title("Steel sample dimensions")
    else:
        plt.title("Yoke dimensions")
```

```
    ax.table(cellText=df.values, colLabels=df.columns, loc="center", cellLoc =␣
 ↪"center")
    plt.show()

table(data1, "I")
table(data2, "S")
table(data3, "Y")
```

Iron sample dimensions

width [mm] +-0.01	height [mm] +-0.01	length [mm] +- 0.01
29.98	29.0	101.62
30.18	29.0	101.5
30.0	29.0	101.52

Steel sample dimensions

width [mm] +-0.01	height [mm] +-0.01	length [mm] +- 0.01
28.0	27.56	102.82
27.98	27.62	102.92
28.0	27.5	102.9

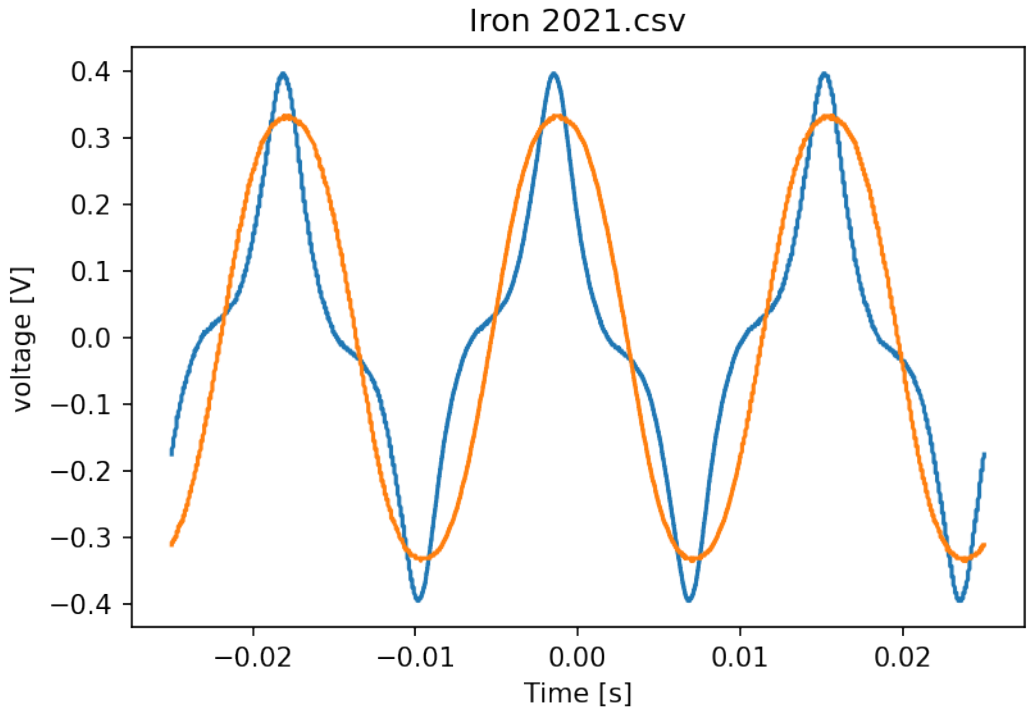
Yoke dimensions

width [mm] +-0.01	height [mm] +-0.01	length [mm] +- 0.01
30.4	150.0	101.7
30.22	150.0	101.9
30.1	150.0	101.98

[167]:
```
def draw(file):
    array_in = np.loadtxt(file, delimiter=',', skiprows = 2)
    x = array_in[:,0] # taking first element of each row (first column)
    ch1 = array_in[:,1] # taking second element of each row (second column)
    ch2 = array_in[:,2] # taking third element of each row (third column)

    max1 = np.argmax(ch1)

    # finding second peak with trial and error
    Period = ch1[max1:1880]
    x1 = x[0]
    x2 = x[Period.size]
    T = x2 - x1
    print(T)
    freq = 1/T
    print(freq)

    plt.plot(x, ch1)
    plt.plot(x, ch2)
    plt.title(file); plt.xlabel('Time [s]'); plt.ylabel('voltage [V]')
    plt.show()
    # showing excerpt of data
    data = np.column_stack((ch1[:5], ch2[:5]))
```

```
    fig, ax = plt.subplots(1,1)
    column_labels = ["Voltage channel 1 [V] +-0.01", "Voltage channel 2 [V] +-0.
 ↪01"]
    df =pd.DataFrame(data, columns=column_labels)
    ax.axis('tight')
    ax.axis('off')
    ax.table(cellText=df.values, colLabels=df.columns, loc="center", cellLoc =␣
 ↪"center")
    plt.title("Excerpt of raw data for Hysteresis " + file)
    plt.show()

    return freq
freq_iron = draw("Iron 2021.csv")
freq_steel = draw("Steel 2021.csv")
```

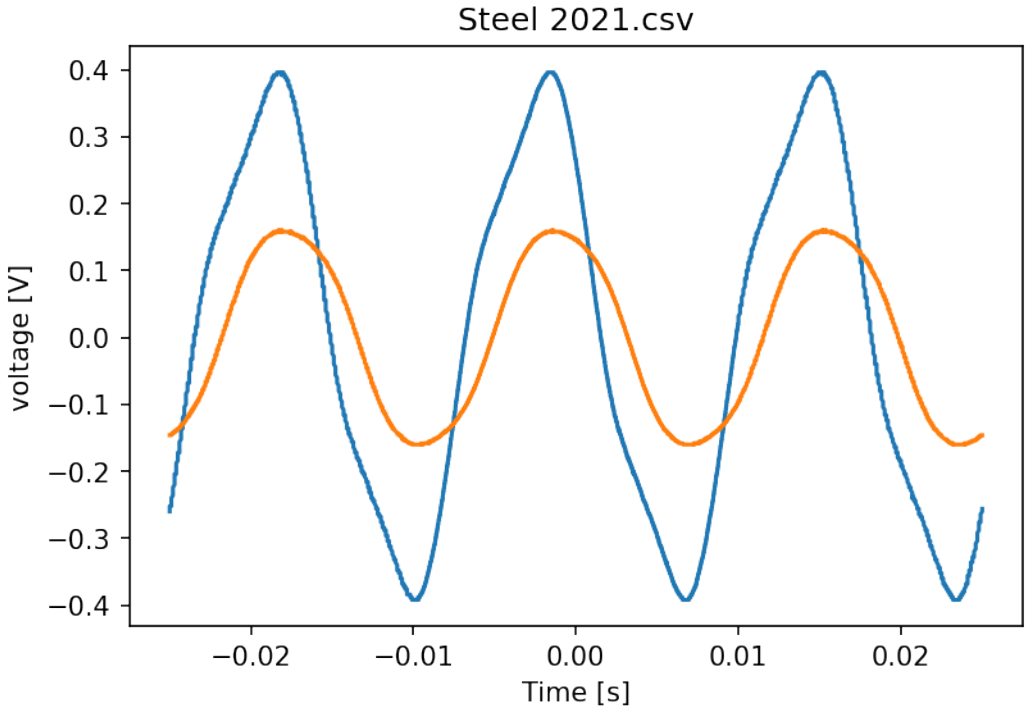
```
0.01674
59.73715651135006
```



Excerpt of raw data for Hysteresis Iron 2021.csv

Voltage channel 1 [V] +-0.01	Voltage channel 2 [V] +-0.01
-0.176	-0.312
-0.172	-0.312
-0.168	-0.308
-0.168	-0.308
-0.164	-0.308

0.016940000000000004
59.03187721369538



Excerpt of raw data for Hysteresis Steel 2021.csv

Voltage channel 1 [V] +-0.01	Voltage channel 2 [V] +-0.01
-0.26	-0.146
-0.256	-0.146
-0.256	-0.146
-0.252	-0.146
-0.252	-0.146

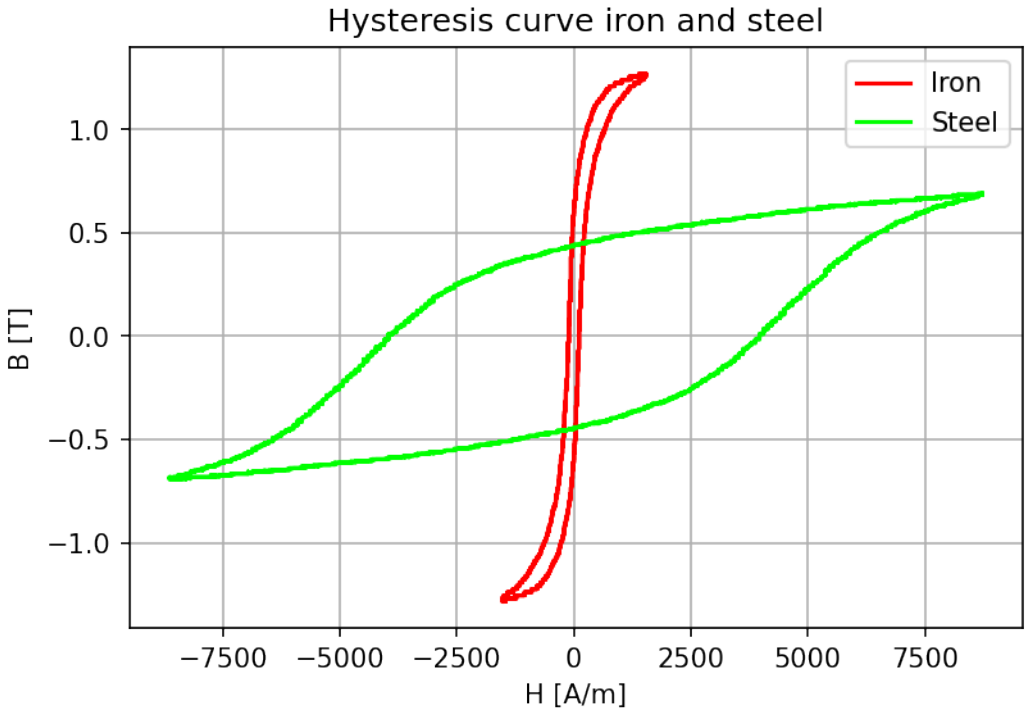
[181]:
```
R = 1.00*1e6 # resistance of resistor in Ohm
R_error = R*(1/100)
C = 0.50/1e6 # Capacitance of capacitor
C_error = C*(2/100)#
S = 0.10 # Resistance of source in Ohm
S_error = S*(5/100)
n1 = 160 # number of turns
n2 = 150 # number of turns

def curve(file):
    array_in = np.loadtxt(file, delimiter=',', skiprows = 2)
    time = array_in[:,0] # taking first element of each row (first column)
    ch1 = array_in[:,1] # taking second element of each row (second column)
    ch2 = array_in[:,2] # taking third element of each row (third column)

    # customizing graph depending on data
    if "Iron" in file:
        Ac = Aci
        Ac_error, L_error = Aci_error, iron_l_error
        L, freq = iron_l+yoke_h*2+yoke_l-yoke_w*3, freq_iron
        L_error = np.sqrt(iron_l_error**2 + yoke_h_error**2 + yoke_l_error**2 +␣
 ↪yoke_w_error**2) # magnetic length
        Farbe, la = 'red', 'Iron'
    else:
        Ac = Acs
        Ac_error, L_error = Acs_error, steel_l_error
```

```
        L, freq = steel_l-yoke_w, freq_steel
        L_error = np.sqrt((steel_l_error)**2 + (yoke_w_error)**2) # megnetic length
        Farbe, la = 'lime', 'Steel'

    B = ((R*C)/(n2*Ac))*ch2 # Calculate B (y axis)
    H = (n1/((L)*S))*ch1 #  Calculate H (x axis)
    plt.plot(H, B, color = Farbe, label = la) # plotting graph
    plt.title("Hysteresis curve iron and steel"); plt.xlabel('H [A/m]'); plt.ylabel('B [T]')
    plt.legend()
    # creating data to then calculate the integral
    # making an array for top part of hysteresis loop and bottom
    # then taking the difference of integrals for area between

    miny = np.amin(B) # finding samllest value
    Bn = B+abs(miny) # shifting data by smallest value so it is aligned with 0 on y-axis

    minpos = np.argmin(Bn)
    maxpos = np.argmax(Bn)
    B_c = Bn[minpos:] # cutting new array off at first minimum position to get start of first full loop

    minpos2 = np.argmin(B_c[1:])
    Bnn = B_c[:np.argmax(B_c)]
    if "Steel" in file:
        minpos2 = np.argmin(B_c[59:]) # extra for Steel because of multiple identical data points
        #print(minpos2)

    Bnn_low = B_c[np.argmax(B_c):minpos2]
    # slicing the arrays to get only one loop top and bottom half
    if "Iron" in file:
        Xup, Yup = H[np.argmin(H):np.argmin(H)+Bnn.size], Bnn
        Xlow, Ylow = H[np.argmin(H)+Bnn_low.size:np.argmin(H)+Bnn_low.size*2], Bnn_low
    else:
        Xup, Yup = H[np.argmin(H):np.argmin(H)+Bnn.size], Bnn
        Xlow, Ylow = H[np.argmin(H)+Bnn.size:np.argmin(H)+Bnn_low.size+Bnn.size], Bnn_low

    # plotting adjusted curves to visualize how well they fit and to see if everything checks out
    #plt.plot(Xup, Yup)
    #plt.plot(Xlow, Ylow)
```

```
    #plt.plot(H, Bn)

    # calculatint the area of top and bottom curve
    Aup, Alow = 0, 0
    for i in range(Xup.size-1):
        dxu = abs(Xup[i+1] - Xup[i]) # getting difference in x values
        dau = dxu * Yup[i] # small change in x multiplied with y value at that position
        Aup += dau # adding small area to total
    for j in range(Xlow.size-1):
        dxl = abs(Xlow[j+1] - Xlow[j])
        dal = dxl * Ylow[j]
        Alow += dal

    area = Alow - Aup # difference for area between
    V = Ac*(L) # volume of magnetic body
    V_error = V*np.sqrt((Ac_error/Ac)**2+(L_error/L)**2+(yoke_w_error/yoke_w)**2)
    print(file, "Results:")
    print("Area of loop:", area, "+- 1%")
    P = area*V*freq # Power loss
    # estimating error in area due to not 100% alignment of area calculcating curves and raw data
    # also estimating error in frequency, again due to jankiness with figuring out peaks and messing with arrays
    P_error = P*np.sqrt((0.01)**2+(V_error/V)**2+(0.005)**2)
    print("Power loss:", P, "+-", P_error, "W")
    # Remanence by finding where H is 0
    remanence_pos = np.where(H == 0)
    remanence = np.average(np.abs(B[remanence_pos]))
    # Coercive force by finding where B is 0
    coercive_force_pos = np.where(B == 0)
    coercive_force = np.average(np.abs(H[coercive_force_pos]))
    # for remanence and coercive force using the error from B and H
    B_error_rel = np.sqrt((R_error/R)**2 + (C_error/C)**2 + (Ac_error/Ac)**2)
    H_error_rel = np.sqrt((L_error/L)**2 + (S_error/S)**2)
    print("Remanence:", remanence, "+-", remanence*B_error_rel, "T")
    print("Coercive force:", coercive_force, "+-", coercive_force*H_error_rel, "A/m")

curve("Iron 2021.csv")
curve("Steel 2021.csv")
plt.grid()
plt.show()
```

```
Iron 2021.csv Results:
Area of loop: 760.3887110451897 +- 1%
```

```
Power loss: 16.337714058144122 +- 0.20060458107188767 W
Remanence: 0.5972523942110577 +- 0.013474042912754796 T
Coercive force: 108.55693585125114 +- 5.428049625785151 A/m
Steel 2021.csv Results:
Area of loop: 8014.633613989763 +- 1%
Power loss: 26.514285714285723 +- 0.32270127812957716 W
Remanence: 0.4428623720278965 +- 0.009935064175801443 T
Coercive force: 3964.7577092511006 +- 198.36608111183764 A/m
```

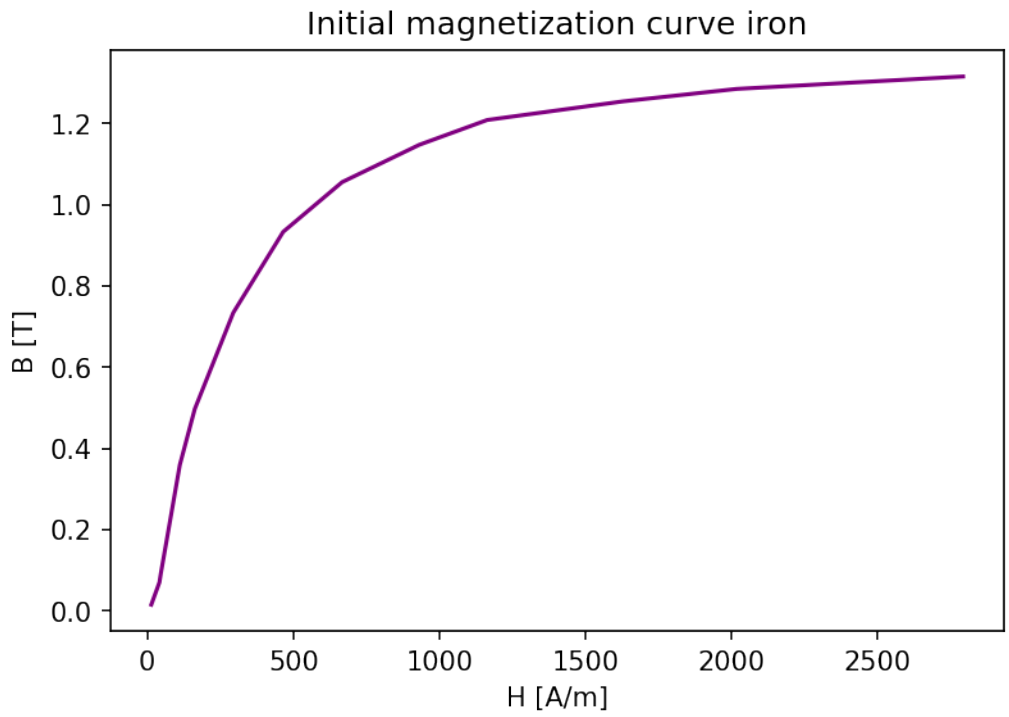


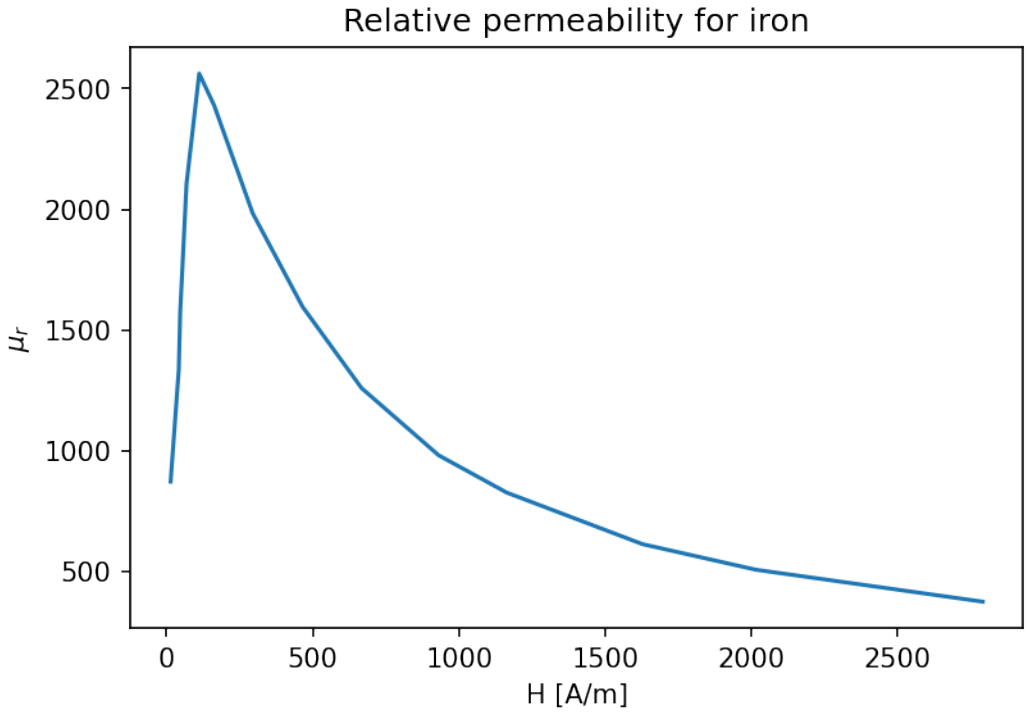
The iron would be better suited than steel for electric motors because the curve shows that for a small increase in H a big increase in B can be achieved and it is therefore also easier to reverse. It also has a smaller area which means less power loss due to internal heat generation. The steel is better for magnetic memory or a permanent magnet because a larger increase in H is needed to change the field which means it is harder to alter the permanent mmagnet which is for obvious reasons desirable.

Comparing power dissipation calculated and the temperature and wattage of a lightbulb the results make sense. The 40W lightbulb was slighty warmer than the iron and steel yoke. So an estimate of around 30W is what I would've guessed for power dissipation. The calculated numbers lie at 26.51 +- 0.32 and 16.34 +- 0.20 for steel and iron. Steel falls close to the inital estimate.

[171]:
```
# initial magnetization curve iron

ch1 = np.array([0.0036, 0.0108, 0.0120, 0.0176, 0.0288, 0.0420, 0.0760, 0.120,␣
↪0.172, 0.240, 0.300, 0.420, 0.520, 0.720])
```

```
ch2 = np.array([0.004, 0.0184, 0.0240, 0.0472, 0.094, 0.130, 0.192, 0.244, 0.
 ↪276, 0.300, 0.316, 0.328, 0.336, 0.344])
# defining variables for iron again
Ac = Aci
Ac_error, L_error = Aci_error, iron_l_error
L, freq = iron_l+yoke_h*2+yoke_l-yoke_w*3, freq_iron

B = ((R*C)/(n2*Ac))*ch2 # Calculate B (y axis)
H = (n1/((L)*S))*ch1 #  Calculate H (x axis)
plt.plot(H, B, color = 'purple') # plotting graph
plt.title("Initial magnetization curve iron"); plt.xlabel('H [A/m]'); plt.
 ↪ylabel('B [T]')
plt.show()

m0 = 1.25663706e-6 # permeability of free space in H/m with error of 1.5e-10 so␣
 ↪negligible for here
mr = B/(H*m0) # no units for relative permeability
# error calculations
B_error_rel = np.sqrt((R_error/R)**2 + (C_error/C)**2 + (Ac_error/Ac)**2)
L_error = np.sqrt(iron_l_error**2 + yoke_h_error**2 + yoke_l_error**2 +␣
 ↪yoke_w_error**2)
H_error_rel = np.sqrt((L_error/L)**2 + (S_error/S)**2)
H_error = H*H_error_rel
mr_error = mr*np.sqrt((H_error_rel)**2 + (B_error_rel)**2)

plt.plot(H, mr)
plt.title("Relative permeability for iron"); plt.xlabel('H [A/m]'); plt.
 ↪ylabel('$\mu_r$')
plt.show()
# getting max permeability and H where it occurs
mr_max = np.amax(mr)
mr_max_pos = np.argmax(mr)
density = H[mr_max_pos]
print("Maximum relative permeability: ", mr_max, "+-", mr_error[mr_max_pos],␣
 ↪"at flux density:", density, "+-", H_error[mr_max_pos], "A/m")
```


Initial magnetization curve iron
B [T]
H [A/m]
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
500
1000
1500
2000
2500


Relative permeability for iron
μ_r
H [A/m]
500
1000
1500
2000
2500
0
500
1000
1500
2000
2500

```
Maximum relative permeability:  2562.206073841252 +- 140.55148649963309 at flux
density: 111.65856258985832 +- 5.5831367579504425 A/m
```

Accepted relative permeability value for 99.8% pure iron is 5000 according to "engineering toolbox" (https://www.engineeringtoolbox.com/permeability-d_1923.html).

[161]:
```
# table for initial magnetization values
data1 = np.column_stack((ch1, ch2))
fig, ax = plt.subplots(1,1)
column_labels = ["X-axis voltage [V] +-0.0002", "Y-axis voltage [V] +-0.0002"]
df = pd.DataFrame(data1, columns=column_labels)
ax.axis('tight')
ax.axis('off')
ax.table(cellText=df.values, colLabels=df.columns, loc="center", cellLoc =␣
 ↪"center")
plt.title("Initial magnetization raw data for iron")
plt.show()
```

Initial magnetization raw data for iron

X-axis voltage [V] +-0.0002	Y-axis voltage [V] +-0.0002
0.0036	0.004
0.0108	0.0184
0.012	0.024
0.0176	0.0472
0.0288	0.094
0.042	0.13
0.076	0.192
0.12	0.244
0.172	0.276
0.24	0.3
0.3	0.316
0.42	0.328
0.52	0.336
0.72	0.344