11/06/20 01:58:19 /home/ana/Documents/uni/PHS3000/code/log4/magnetic.py

```python
# PHS3000
# Magnetic susceptibility
# Ana Fabela, 29/10/2020
import os
from pathlib import Path
import csv
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
from physunits import *

plt.rcParams['figure.dpi'] = 150
read_folder = Path('data')
np.seterr(divide='ignore', invalid='ignore')

# Globals
# c = 299792458 # [m/s]
π = np.pi
hbar = 1.0545718e-34 * J * s
e_charge = 1.60217662e-19 * C
e_mass = 9.10938356e-31 * kg
gprime = 9.8 * m / s**2
μ0 = 1.25663706212e-6 * H / m # vacuum permeability
k = 1.38064852e-23 * m**2 * kg /(s**2 * K) # Boltzmann constant
β = e_charge * hbar / (2 * π * e_mass)

N_A = 6.0221409e+23 # Avogadro's number

# WIKIPEDIA VALUES
# hexahydrate values
M_Ni = 262.85 * g / N_A
ρ_Ni = 2.07 * g / cm**3
N_Ni = ρ_Ni / M_Ni
# tetrahydrate values
M_Mn = 223.07 * g / N_A
ρ_Mn = 1339 * g / cm**3
N_Mn = ρ_Mn / M_Mn
# hexahydrate values
M_Co = 263.08 * g / N_A
ρ_Co = 2.019 * g / cm**3
N_Co = ρ_Co / M_Co

g_S = 2
Temps = 294 * K
u_Temps = 1 * K
Area = 38.5e-6 * m**2
u_Area = 0.05e-6 * m**2

```

```python

# from eyeballing calibration data
linear_guess_0 = [0.3, 0] # [gradient, intercept]

# from eyeballing linearised magnetic sample data (I**2)
linear_guess_2 = [2e-6, 0] # [gradient, intercept]

# from eyeballing delta_m = Dprime*B**2 for magnetic sample data
linear_guess_3 = 2e-3 # gradient

def load_data(filename):
    xs = []
    ys = []
    for line in filename:
        x, y = line.split(',')
        xs.append(float(x))
        ys.append(float(y))
    return np.array(xs), np.array(ys)

def line_fit(x, m, c):
    return m * x + c

def fitting_calibration_data(xs, ys, initial_guess):
    pars, pcov = scipy.optimize.curve_fit(line_fit, xs, ys, p0=initial_guess)
    perr = np.sqrt(np.diag(pcov))
    # print(f"{pars}, {perr}\n")
    linear_fit = line_fit(xs, *pars)
    return pars, perr, linear_fit

def plot_data(xs, ys, u_xs, u_ys, fit, filename, field=False, squared=False, calibration=False):
    if calibration:
        plt.plot(xs, fit, color='teal',label=r"fit")
        plt.errorbar(xs, ys,
            xerr=u_xs,yerr=u_ys,color='orange',
            marker='None',linestyle='None', label="Magnetic field"
        )
        plt.ylabel('Magnetic field / T')
        plt.xlabel("Current / A")

    elif squared:
        # plot of data and fit of equation (19)
        plt.errorbar(xs, ys,
                    xerr=u_xs, yerr=u_ys, color='indigo',
                    marker='None',linestyle='None', label="Mass"
        )
        plt.ylabel('Mass / kg')

        if field:
```

```python
            plt.plot(xs, fit, color='lavender',label=r"fit")
            plt.xlabel(r"Magnetic field squared / $T^2$")
        else:
            plt.plot(xs, fit, color='plum',label=r"fit")
            plt.xlabel(r"Current squared / $A^2$")

    else:
        plt.errorbar(xs, ys,
            xerr=u_xs, yerr=u_ys, color='olive',
            marker='None',linestyle='None', label="Mass"
        )
        plt.ylabel('Mass / kg')
        plt.xlabel("Current / A")

    plt.title(f"{filename}")
    plt.legend()
    plt.show()


def propagate_uncertainty(i, xs, ys, calibration=False):
    u_xs = []
    u_ys = []
    # print("New sample")
    for x in xs:
        u_x = 0.012 * x
        u_xs.append(u_x)
        # print(f"{u_x = }")
    if calibration:
        # uncertainty in Magnetic field measurements: using typical uncertainty
        for y in ys:
            u_y = 0.050 * y + 0.020 * 300 # [mT]
            # print(f"{u_y = }")
            u_ys.append(u_y)
    else:
        # uncertainty in mass measurements: using repeatability and linearity
        for y in ys:
            # print("New sample")
            u_y = np.sqrt(0.0001**2 + (0.0002/120 * y)**2 + 0.00005**2) # [g]
            # print(f"{u_y = }")
            u_ys.append(u_y)
    return np.array(u_xs), np.array(u_ys)

def equation_19(B, Dprime):
    return Dprime * B

def fitting_data(B_squared, ys, initial_guess):
```

```
    Dprime, pcov = scipy.optimize.curve_fit(equation_19, B_squared, ys, p0=initial_guess)
    perr = np.sqrt(np.diag(pcov))
    # print(f"{Dprime}, {perr}\n")
    eqn_19_fit = equation_19(B_squared, Dprime)
    return Dprime, perr, eqn_19_fit

def main():
    files = list(os.listdir(path=read_folder))
    files.sort(key=lambda name: int(name.strip('.csv').split('_')[-1]) if name[-5].isdigit() else -1)
    # print(files)
    file_names = []

    for i, file in enumerate(files):
        name = file.split(".")[0]
        file_names.append(name)
        # print(name)
        # print(i, file)
        file = open(read_folder / file)
        xs, ys = load_data(file)
        if i == 0:
            u_xs, u_ys = propagate_uncertainty(i, xs, ys, calibration=True)
            ys = ys * mT # convert to Teslas
            u_ys = u_ys * mT # convert to Teslas
            pars0, perr0, fit0 = fitting_calibration_data(xs, ys, linear_guess_0)
            # plot_data(xs, ys, u_xs, u_ys, fit0, name , field=False, squared=False, calibration=True)
            print(f"\nCalibration data:\nI={xs} ± {u_xs}\nB={ys} ± {u_ys}")
            print(f"\nfit parameters for calibration:\n    m = {pars0[0]:.4f} ± {perr0[0]:.4f}\n    c = {pars0[1]:.4f} ± {perr0[1]:.4f}")

        elif i == 1:
            u_xs, u_ys = propagate_uncertainty(i, xs, ys, calibration=False)
            dummy_ys = ys
            u_dummy_ys = u_ys
            kg_ys = ys / 1000
            u_kg_ys = u_ys / 1000
            print(f"\nSample {i}: I=\n{xs} ± {u_xs}\nm={ys} ± {u_ys}")
            # plot_data(xs, kg_ys, u_xs, u_kg_ys, fit0, name , field=False, squared=False, calibration=False)


```

```python
        elif i > 1:
            u_xs, u_ys = propagate_uncertainty(i, xs, ys, calibration=False)
            corrected_ys = (ys - dummy_ys) # to correct for dummy values. Units: [g]

            total_ys = corrected_ys / 1000 # convert to kg
            u_total_ys = np.sqrt(u_ys**2 + u_dummy_ys**2) / 1000 # Units: [kg]
            # print(f"\nSample {i}:\nu(I)={xs} ± {u_xs}\nu(m)={u_total_ys}")

            # quadratic plot
            # plot_data(xs, total_ys, u_xs, u_total_ys, fit0, name, field=False, squared=False, calibration=False)

            u_xs_squared = []
            for x in xs:
                u_x_squared = 2 * 0.012 * x**2
                u_xs_squared.append(u_x_squared)

            # current squared - Linearised plot and fit
            pars2, perr2, fit2 = fitting_calibration_data(xs**2, total_ys, linear_guess_2)
            # plot_data(xs**2, total_ys, u_xs_squared, u_total_ys, fit2, name, field=False, squared=True, calibration=False)

            # print(f"Current squared u(I**2)={u_xs_squared}")
            print(f"\nfit parameters for sample {i}:\n    m = {pars2[0] * 1e6:.4f} ± {perr2[0] * 1e6:.4f} * 1e-6\n    c = {pars2[1]* 1e6:.4f} ± {perr2[1]* 1e6:.4f} * 1e-6")

            # calculating B and propagating uncertainty
            mI = pars0[0] * xs
            u_mI = np.sqrt((perr0[0] / pars0[0])**2 + (u_xs/ xs)**2) * mI
            B = mI + pars0[1]
            u_B = np.sqrt(u_mI**2 + perr0[1]**2)

             # squaring B and propagating uncertainty
            B_squared = (mI + pars0[1])**2
            u_B_squared = 2* u_B * B
            # print(f"Magnetic field squared u(B**2)={u_B_squared}")

            # fiting equation (19)
            Dprime, perr, eqn_19_fit = fitting_data(B_squared, total_ys, linear_guess_3)
            print(f" Equation (19) gradient:\n    D' = {Dprime[0]:.6f} ± {perr[0]:.6f}")
            χ = 2 * μ0 * gprime * Dprime[0] / Area
```

```
            u_χ = χ * np.sqrt((perr[0] / Dprime[0])**2 + (u_Area / Area)**2)
            print(f"\nMagnetic susceptibility for sample {i}:\n{χ = :.6f} ± {u_χ:.6f}")

            # plot of data and fit of equation (19)
            # plot_data(B_squared, total_ys, u_B_squared, u_total_ys, eqn_19_fit, name, field=True, squared=True, calibration=False)
            if i == 2:
                # print(f"{M_Ni=}\n{ρ_Ni=}\n{N_Ni=}\n")
                RHS = 6 * gprime  * k * Temps * Dprime / (Area * N_Ni)
                print(f"\nSample {i}g^2 * β^2 * J(J+1) = {RHS[0]}")
                SSplus1 = RHS / (4 * β**2)
                u_SSplus1 = np.sqrt((u_Temps / Temps)**2 + (u_Area / Area)**2 + (perr[0]/Dprime)**2) * SSplus1
                S_plus = (-1 + np.sqrt(1 + 4 * SSplus1 )) / 2
                S_minus = (-1 - np.sqrt(1 + 4 * SSplus1 )) / 2
                u_S_plus = (1/2 * u_SSplus1 / SSplus1) * S_plus
                u_S_minus = (1/2 * u_SSplus1 / SSplus1) * S_minus

                print(f"S = {S_plus} ± {u_S_plus} or \n    {S_minus} ± {u_S_minus}")

            elif i == 3:
                # print(f"{M_Mn=}\n{ρ_Mn=}\n{N_Mn=}\n")
                RHS = 6 * gprime  * k * Temps * Dprime / (Area * N_Mn)
                print(f"\nSample {i} g^2 * β^2 * J(J+1) = {RHS[0]}")
                SSplus1 = RHS / (4 * β**2)
                u_SSplus1 = np.sqrt((u_Temps / Temps)**2 + (u_Area / Area)**2 + (perr[0] / Dprime)**2) * SSplus1
                S_plus = (-1 + np.sqrt(1 + 4 * SSplus1 )) / 2
                S_minus = (-1 - np.sqrt(1 + 4 * SSplus1 )) / 2
                u_S_plus = (1/2 * u_SSplus1 / SSplus1) * S_plus
                u_S_minus = (1/2 * u_SSplus1 / SSplus1) * S_minus

                print(f"S = {S_plus} ± {u_S_plus} or \n    {S_minus} ± {u_S_minus}")

            elif i == 5:
                print(f"{M_Co=}\n{ρ_Co=}\n{N_Co=}\n")
                w = 23.0 * (μ0 * N_Co * β**2 / (3 * k * χ))
                u_w = w * np.sqrt((u_Area / Area)**2 + (perr[0] / Dprime)**2)
                Θ = Temps - w
                u_Θ = np.sqrt(u_Temps**2 + u_w**2)
                print(f"\nSample {i} w = {w} ± {u_w} K")
                print(f"\nWeiss temperature Θ = {Θ} ± {u_Θ} K\n")
```

```
                    assert(0)


main()

```