Electron Spin Resonance

PHY224 Lab 6

Fredrik Dahl Bråten, Pankaj Patil

November 11, 2021

1 Abstract

In this exercise we experimentally calculate the gyromagnetic ratio γ for an electron. As we expect the ratio to be greater than the theoretically predicted value of $e/2m$, we compute this discrepancy in terms of *Lande g factor*. This analysis was done in Python by use of the numpy, scipy and matplotlib modules.

2 Introduction

The potential energy acquired by an electron in presence of external magnetic field $\vec{B}$ is given by

$$E = -\vec{\mu} \cdot \vec{B}$$

Where, μ is magnetic dipole moment of the electron.

The relationship between magnetic moment and the spin angular momentum of the electron is given by,

$$\vec{\mu} = \gamma \vec{S}$$

Where γ is gyromagnetic ratio.

For ideal electron, we expect γ to be equal to $e/2m$. But for real world electron, it it greater than $e/2m$. This discrepancy is given by *Lande g Factor*, which is defined as

$$g = \frac{\gamma}{e/2m} > 1$$

If an electron absorbs photon with energy equal to the difference between the two spin energy level, then it can flip it's orientation. This phenomenon is called *Electron Spin Resonance*.

The energy difference between the two spin levels is given by,

$$\Delta E = \gamma \hbar B_z$$

Where, B_z is the uniform z-component of the external magnetic field. By equating above with energy of a photon $h\nu$, we get

$$\nu = \frac{1}{2\pi}\gamma B_z$$

Now in the experimental setup the external magnetic field is generated by the two Helmholtz coils. The magnetic field produced by the coils is given by,

$$B_z = \left(\frac{4}{5}\right)^{\frac{3}{2}} \frac{\mu_0 n I}{R}\ T$$

Where,

$\mu_0 = 4\pi \times 10^{-7}\ N/A^2$

n = number of turns = 320

I = current through thee coils

R = the radius of the coils

By measuring the current using the oscilloscope, we compute the magnetic field produced by the coils. Using the frequency counter, note down the resonance frequency. With these values we can use the relationship between ν and B_z to compute γ and hence the *Lande g Factor*.

3 Methods, Materials and Experimental Procedure

We successfully followed the procedures as described by the TA and lab manual [1] for this experiment.

4 Results

From our model of Frequency vs Magnetic field we found the linear relationship between the two is given by

$$f(B) = m\nu$$

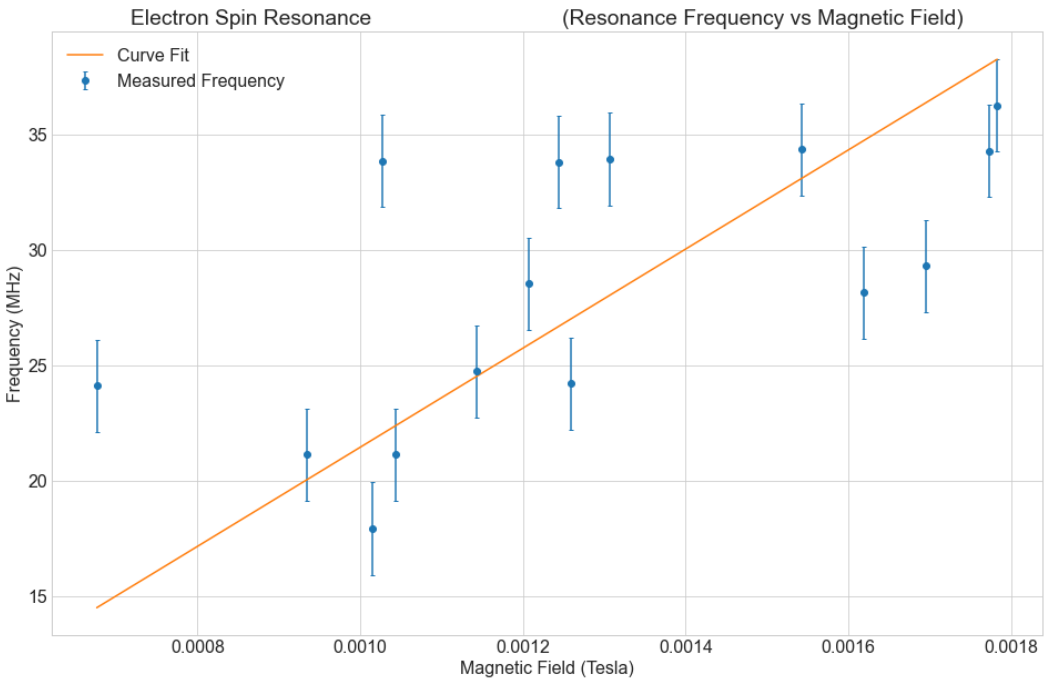
Where,

B = External Magnetic Field in Tesla

ν = Resonance Frequency in MHz

m = Slope of the linear curve

This relation is plotted in following graph,



Model Function : $f(x) = mx$
Slope $m = 21471.7312\ T/MHz$
$\chi^2_{red} = 8.2622$

Figure 1: Frequency vs Magnetic Field

After the data analysis using Python, we found

$m = 21471.7312\ T/MHz$

Gyromagnetic Ratio $\gamma = 134910.87 \pm 2450.10\ rad/s/T$

Lande g Factor $= 1.53 \pm 0.03$

$\chi^2_{red} = 8.2622$

Using the measured currents, which give us the measured external magnetic field, and the measured resonance frequency, we obtained *Lande g Factor* for each measurement. These factors for each frequency is plotted in following graph. The standard deviation of the Lande g Factors is used for the error bars in the graph.

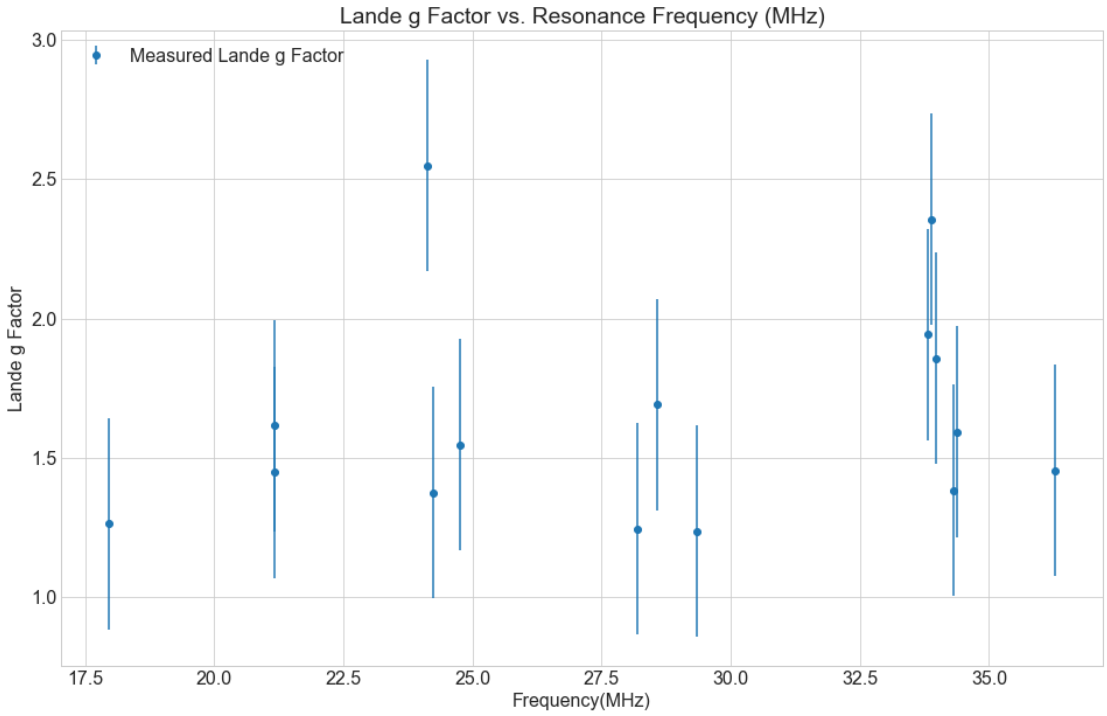


Figure 2: Lande g Factor vs Frequency

5 Discussion

For the linear model without the y-intercept, we found the linear fit with $\chi^2_{red} = 8.2622$. As this value is in order of 1, we can say that the linear fit is a good fit.

From the Figure 1, it is clear that not all the data points are close to the linear fit. These large deviations can be attributed to sensitivity of the instruments, and also large uncertainty in the measurements.

We observed that the copper coils used to put electron in precision have lot of impact on the data. The data points are closer to the line for the copper coil with highest number of turns. For medium number of turns, the data points are further away from the linear fit. We did not get any good measurement for the copper coil with lowest number of turns. It was difficult to achieve precise resonance frequency for the coil with lowest number of turns.

We observed very high uncertainty (2 MHz) in the measurement of resonance frequency.

Now we answer the questions asked in the manual

Question 1 The ESR signal is no symmetric about the maximum.

Question 2 The relation between resonance frequency and magnetic field is not always linear.

Question 3 The width of the peak ...

6 Conclusions

In this exercise we studied the Electron Spin Resonance phenomenon. Using the uniform external magnetic field, we observe the variation of resonance frequency with respect to the current through the Helmholtz coils producing the magnetic field. As predicted by theory, we observe the relationship is linear, and the linear fit to the measured data was found to be in order of magnitude 1. Using the linear model fit, we computed the gyromagnetic ratio γ and the *Lande g Factor*. Again as predicted by the theory the value of *Lande g Factor* was found to be greater than 1.

A Appendix

A.1 Oscilloscope Readings

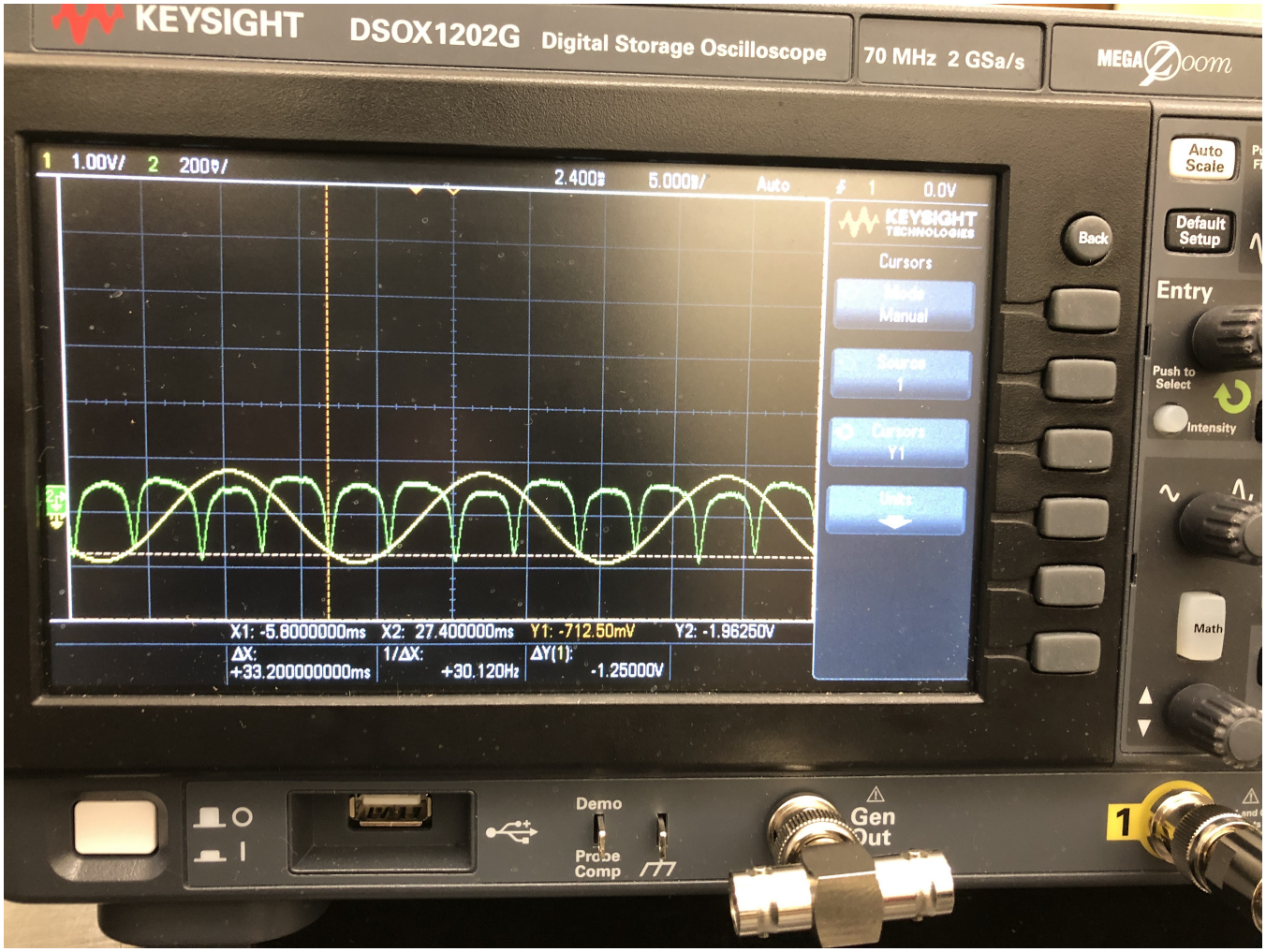


Figure 3: Oscilloscope Reading

A.2 Python Code

The Python code for this exercise is divided into two files. statslab.py file contains utility methods which we will be frequently using in this course. lab_6.py file contains the code which analyzes the data.

A.2.1 statslab.py

```
import numpy as np
import scipy.optimize as optim
import matplotlib.pyplot as plt


##########################################################################
# Utility Methods Library
#
# This file contains some utility method which are common to our data analysis.
# This library also contains customized plotting methods.
##########################################################################

# use bigger font size for plots
plt.rcParams.update({'font.size': 16})

def chi2(y_measure,y_predict,errors):
    """Calculate the chi squared value given a measurement with errors and
    prediction"""
    return np.sum( np.power(y_measure - y_predict, 2) / np.power(errors, 2) )

def chi2reduced(y_measure, y_predict, errors, number_of_parameters):
    """Calculate the reduced chi squared value given a measurement with errors
    and prediction, and knowing the number of parameters in the model."""
    return chi2(y_measure, y_predict, errors)/ \
            (y_measure.size - number_of_parameters)

def read_data(filename, skiprows=1, usecols=(0,1), delimiter=","):
    """Load give\n file as csv with given parameters,
    returns the unpacked values"""
    return np.loadtxt(filename,
                      skiprows=skiprows,
                      usecols=usecols,
                      delimiter=delimiter,
                      unpack=True)
```

```
def fit_data(model_func, xdata, ydata, yerrors, guess=None):
    """Utility function to call curve_fit given x and y data with errors"""
    popt, pcov = optim.curve_fit(model_func,
                                 xdata,
                                 ydata,
                                 absolute_sigma=True,
                                 sigma=yerrors,
                                 p0=guess)

    pstd = np.sqrt(np.diag(pcov))
    return popt, pstd

# y = ax+b
def linear_regression(xdata, ydata):
    """Simple linear regression model"""
    x_bar = np.average(xdata)
    y_bar = np.average(ydata)
    a_hat = np.sum( (xdata - x_bar) * (ydata - y_bar) ) / \
            np.sum( np.power((xdata - x_bar), 2) )
    b_hat = y_bar - a_hat  * x_bar
    return a_hat, b_hat

class plot_details:
    """Utility class to store information about plots"""
    def __init__(self, title):
        self.title = title
        self.x_log_scale = False
        self.y_log_scale = False

    def errorbar_legend(self, v):
        self.errorbar_legend = v
    def fitted_curve_legend(self, v):
        self.fitted_curve_legend = v
    def x_axis_label(self, v):
        self.x_axis_label = v
    def y_axis_label(self, v):
        self.y_axis_label = v
    def xdata(self, x):
        self.xdata = x
    def ydata(self, y):
        self.ydata = y
    def yerrors(self, y):
        self.yerrors = y
    def xdata_for_prediction(self, x):
        self.xdata_for_prediction = x
    def ydata_predicted(self, y):
        self.ydata_predicted = y
```

```
    def legend_position(self, p):
        self.legend_loc = p
    def chi2_reduced(self, c):
        self.chi2_reduced = c
    def set_x_log_scale(self, c):
        self.x_log_scale = c
    def set_y_log_scale(self, c):
        self.y_log_scale = c

def plot(plot_details, new_figure=True, error_plot=True):
    """Utility method to plot errorbar and line chart together,
    with given arguments"""
    if new_figure:
        fig  = plt.figure(figsize=(16, 10))
        fig.tight_layout()
        plt.style.use("seaborn-whitegrid")

    # plot the error bar chart
    if error_plot:
        plt.errorbar(plot_details.xdata,
                     plot_details.ydata,
                     yerr=plot_details.yerrors,
                     marker="o",
                     label=plot_details.errorbar_legend,
                     capsize=2,
                     ls="")

    # plot the fitted curve
    plt.plot(plot_details.xdata_for_prediction,
             plot_details.ydata_predicted,
             label=plot_details.fitted_curve_legend)

    # legend and title
    plt.title(plot_details.title)
    plt.xlabel(plot_details.x_axis_label)
    plt.ylabel(plot_details.y_axis_label)

    if plot_details.x_log_scale:
        plt.xscale("log")

    if plot_details.y_log_scale:
        plt.yscale("log")

    legend_pos = "upper left"
    if hasattr(plot_details, "legend_loc"):
        legend_pos = plot_details.legend_loc
```

```
    plt.legend(loc=legend_pos)
```

A.2.2 lab_6.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math
import statslab as utils
import matplotlib.pyplot as plt
import numpy as np


# constants for this exercise

# constant uncertainity in frequency measurement in MHz
frequency_uncertainity = 2

# electron charge in Coulomb
e = 1.602 * 10 ** (-19)

# electron mass ini kg
m = 9.109 * 10 ** (-31)

# permeability
mu = 4*math.pi* (10**(-7))

# number of turns of the external coils
n = 320

# radius of the coils in mm
R = 70

# linear model function without y-intercept
def model_function(currrent, slope):
    return currrent * slope

# import the measured data from data files
files = ["../../data/Highest_N_Data.csv", "../../data/Medium_N_Data.csv"]

measured_currents = np.array([])
measured_frequency = np.array([])

# iterate over all the files to collect measured currrents and frequency
for f in files:
    c, f = utils.read_data(f,
                           usecols=(2, 1),
```

```
                            skiprows=1)

    measured_currents = np.append(measured_currents, c)
    measured_frequency = np.append(measured_frequency, f)

# measure current correction as the current is split between two coils.
measured_currents = measured_currents/2

# frequency errors
frequency_errors = np.ones_like(measured_frequency) * frequency_uncertainity

# compute the measured magnetic field in Tesla
measured_magnetic_field = (4/5)**(3/2) * mu * n * measured_currents / R

# compute the gyromagnetic ratio for each measurement
gamma_measured = 2*math.pi*measured_frequency/measured_magnetic_field

# compute the Lande g factor
g_measured = gamma_measured / (e / (2 * m)) * 10 ** 6

# standard devation in g factor
std_g = np.std(g_measured)
print("Standard deviation of measured Lande g Factor = %.2f" % std_g)


# fit the data to our model function
popt, pstd = utils.fit_data(model_function,
                            measured_magnetic_field,
                            measured_frequency,
                            frequency_errors)

slope = popt[0]

print("Slope of the line (frequency vs magnetic field) = %.4f" % slope)

# compute gamma (gyromagnetic  ratio) using fit data
gamma = 2*math.pi*slope
gamma_var = 2*math.pi* pstd[0]
print("gamma (gyromagnetic  ratio) = %.2f +/- %.2f rad/s/T" % (gamma, gamma_var))

# compute Lange g Factor
g = gamma / (e / (2 * m)) * 10**6
gvar = gamma_var / (e / (2 * m)) * 10**6
print("Lande g Factor= %.2f +/- %.2f" % (g, gvar))

# prepare data for prediction using the curve fit slope value
m_min = np.min(measured_magnetic_field)
```

```
m_max = np.max(measured_magnetic_field)
magnetic_field_for_prediction = np.linspace(m_min, m_max, 1000)

# compute the predicted frequencies
predicted_fequency = model_function(magnetic_field_for_prediction,
                                    slope)

# compute chi2r
chi2r_curve_fit = utils.chi2reduced(measured_frequency,
                                    model_function(measured_magnetic_field,
                                                   slope),
                                    frequency_errors,
                                    1)

print("chi2reduced = %.4f" % chi2r_curve_fit)

# plot measured frequency vs magnetic field, with the curve fit
plot_data = utils.plot_details("Electron Spin Resonance \
                               (Resonance Frequency vs Magnetic Field)")
plot_data.errorbar_legend("Measured Frequency")
plot_data.fitted_curve_legend("Curve Fit")
plot_data.x_axis_label("Magnetic Field (Tesla)")
plot_data.y_axis_label("Frequency (MHz)")
plot_data.xdata(measured_magnetic_field)
plot_data.ydata(measured_frequency)
plot_data.yerrors(frequency_errors)
plot_data.xdata_for_prediction(magnetic_field_for_prediction)
plot_data.ydata_predicted(predicted_fequency)
plot_data.legend_position("upper left")
plot_data.chi2_reduced(chi2r_curve_fit)

# plot the data
utils.plot(plot_data)

# save the plot
plt.savefig("lab6_freq_vs_magnetic_field.png")

fig  = plt.figure(figsize=(16, 10))
fig.tight_layout()
plt.style.use("seaborn-whitegrid")
plt.errorbar(measured_frequency, g_measured, marker="o",
             yerr=std_g*np.ones_like(g_measured), fmt=" ",
             label="Measured Lande g Factor")
plt.xlabel("Frequency(MHz)")
plt.ylabel("Lande g Factor")
plt.title("Lande g Factor vs. Resonance Frequency (MHz)")
plt.legend(loc="upper left")
```

```
# save the plot
plt.savefig("lab6_g_factor_vs_freq.png")
```

References

[1] Electron Spin Resonance - electric-spin.pdf.pdf