Week 12 – Week commencing 9/5/2022

**Week Outline**

Online meeting. Since final report due date is the 13th May, this meeting focuses on discussing the draft final report and corrections. Main problems of the report included:

* Over word limit
* Terms and symbols that are introduced not defined
* Not succinct

Recommendations:

* Removing sections that are not as consequential to lower word count (e.g. interference, excess noise dF0 data that masks hot bar response)
* More clarification on all introduced concepts and symbols
* Add diagram for airport security camera image, showing function
* Shorten language in exchange for succinctness. (e.g. As seen previously in Figure 3… -> Figure 3 shows…)

**Outline of Tasks**

* Correct the final report based on recommendations and advice given above.

**Complete Python Code For Whole Project**

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

from scipy.special import iv as I0

from scipy.special import kv as K0

import fsab\_dirfile\_raw as fsab

import os

from scipy import signal

from scipy import constants as const

from scipy.interpolate import UnivariateSpline

import csv as csv

from matplotlib.lines import Line2D

from matplotlib import cm

#Constant Experiment variables

T\_Hotbar = 41.3

T\_Surrounding = 22.7

dT = T\_Hotbar - T\_Surrounding

T\_Hotbar\_K = T\_Hotbar + 273

T\_Surrounding\_K = T\_Surrounding + 273

#Hard Code User Inputs

data\_folder\_name = "ch1"

KID\_Number = 2

lower\_range = 1.5725

upper\_range = 1.5875

def main():

    global KID\_Number

    global lower\_range

    global upper\_range

    #Reading data

    local\_file = os.getcwd()

    data = fsab.fsab\_dirfile(local\_file + "\\"  + data\_folder\_name)

    #Plotting Sweep

    user\_input = input("Default analysis (y/n)? (KID = 2, 1.5725 < hotbar < 1.5875)\n")

    #Get vals

    if user\_input.lower() == "n":

        KID\_Number, upper\_range, lower\_range = get\_user\_input(data)

    I\_Time, Q\_Time, time = Get\_IQ\_Time(data, KID\_Number)

    plt.plot(time, np.sqrt(I\_Time\*\*2+Q\_Time\*\*2), color='black')

    plt.ylabel("|S21| / V")

    plt.xlabel("time / s")

    plt.show()

    plt.figure()

    I, Q, sweep, tone\_freq = Get\_IQ\_Sweep(data, KID\_Number)

    print(tone\_freq)

    plt.plot(sweep/1e9, np.sqrt(I\*\*2+Q\*\*2), color='black')

    plt.ylabel("|S21| / V")

    plt.xlabel("Frequency / GHz")

    plt.show()

    plt.figure()

    dF0, time = Get\_dF0(data, KID\_Number)

    plt.plot(time, dF0, color='black')

    plt.ylabel("dF0 / Hz")

    plt.xlabel("time / s")

    plt.ticklabel\_format(useOffset=False)

    plt.show()

    plt.figure()

    #Get range of time for peak height

    peak\_range\_array = np.where(np.logical\_and(time>=lower\_range, time<=upper\_range))

    #Get index of upper and lower range for time

    lower\_index = peak\_range\_array[0][0]

    upper\_index = peak\_range\_array[0][-1]

    plt.plot(time[lower\_index-40:upper\_index+40], dF0[lower\_index-40:upper\_index+40], marker="x", markersize=2, color="black", linewidth=0)

    plt.xlabel('Time / s')

    plt.ylabel('dF0 / Hz')

    plt.show()

    plt.figure()

    #Plot this range

    plt.plot(time[lower\_index-40:upper\_index+40], dF0[lower\_index-40:upper\_index+40], marker="x", markersize=2, color="black", linewidth=0, label="Data")

    #Curve fit

    #P\_Guess

    height = 400

    mu = time[upper\_index] - time[lower\_index]

    sigma = 1

    c = 3000

    p\_guess = [height, mu, sigma, c]

    #Gaussian curve

    popt\_main, \_ = curve\_fit(Gaussian, time[lower\_index:upper\_index], dF0[lower\_index:upper\_index], maxfev=20000, p0=p\_guess)

    response\_main = popt\_main[0]/dT

    dF0\_hotbar = popt\_main[0]

    #Plot curve\_fit

    plt.plot(time[lower\_index-30:upper\_index+30], Gaussian(time[lower\_index-30:upper\_index+30], \*popt\_main), label="Curve Fit")

    plt.xlabel('Time / s')

    plt.ylabel('dF0 / Hz')

    plt.legend(loc='upper right', fancybox=True)

    plt.show()

    print(\*popt\_main)

    responsivity = list()

    #Loop for all KIDs

    number\_of\_KID = data.numkids

    #P\_Guess

    height = 400

    mu = time[upper\_index] - time[lower\_index]

    sigma = 1

    c = 3000

    p\_guess = [height, mu, sigma, c]

    #popt=[]

    # for KID\_Num in range(0, number\_of\_KID):

    #     dF0, time = Get\_dF0(data, KID\_Num)

    #     #Get index of upper and lower range for time

    #     lower\_index = peak\_range\_array[0][0]

    #     upper\_index = peak\_range\_array[0][-1]

    #     #Curve Fit

    #     try:

    #         popt, \_ = curve\_fit(Gaussian, time[lower\_index:upper\_index], dF0[lower\_index:upper\_index], maxfev=10000, p0=p\_guess)

    #         #Get curve heigh

    #         dF0\_hotbar = popt[0]

    #         #Get response

    #         response = dF0\_hotbar/dT

    #         if abs(response) <= 100:

    #             responsivity.append(response)

    #         else:

    #             responsivity.append(0)

    #     except RuntimeError:

    #         responsivity.append(0)

    # plt.plot(responsivity, marker='x', color='black')

    # plt.title("Responsivities of all KIDs")

    # plt.ylabel("Responsivity / Hz K^-1")

    # plt.xlabel("KID Number")

    # plt.ticklabel\_format(useOffset=False)

    # plt.show()

    # plt.figure()

    #Spectral\_densities

    frequencies, spectral\_densities = fourier\_transform(data, KID\_Number)

    plt.plot(frequencies, np.sqrt(spectral\_densities), color='black')

    plt.semilogy()

    plt.title("Noise Spectral Densities vs Frequencies")

    plt.ylabel("Spectral Densities / WHz$^{-0.5}$")

    plt.xlabel("Frequency / Hz")

    plt.show()

    plt.figure()

    #NET

    Noise\_Eq\_T = np.sqrt(spectral\_densities)/response\_main

    plt.loglog(frequencies[1:], (Noise\_Eq\_T[1:]), color="black")

    plt.title("NET vs frequencies for KID Number " + str(KID\_Number))

    plt.ylabel("NET / KHz$^{-0.5}$")

    plt.xlabel("Frequency / Hz")

    plt.show()

    plt.figure()

    #Get Transmission Factor

    transmission\_frequency, transmission = np.loadtxt('MUSCAT\_band.txt', unpack=True)

    #Get dnu using fwhm

    fwhm, midpoint\_frequency, lower\_bandwidth, upper\_bandwidth = transmission\_fwhm(transmission\_frequency, transmission)

    print("BANDWIDTH")

    print(upper\_bandwidth)

    print(lower\_bandwidth)

    dnu = upper\_bandwidth-lower\_bandwidth

    print(dnu)

    wavelength = const.c/midpoint\_frequency

    print("fwhm")

    print(fwhm)

    print("midpoint")

    print(midpoint\_frequency)

    #Blackbody Intensities

    room\_power = get\_power(planck, transmission\_frequency, transmission, wavelength, dnu, BFF=1, T=T\_Surrounding\_K)

    #Beam filling factor

    Beam\_Filling\_Factor = calculate\_Beam\_Filling\_Factor(length=100, width=20, FWHM=20, points=len(transmission\_frequency))

    print(Beam\_Filling\_Factor)

    hotbar\_room\_power = get\_power(planck, transmission\_frequency, transmission, wavelength, dnu, BFF=Beam\_Filling\_Factor, T=T\_Hotbar\_K)

    print("Room optical power = " + str(hotbar\_room\_power) + " W")

    print("Room optical power = " + str(room\_power) + " W")

    total\_photon\_noise = np.sqrt(shot\_noise(room\_power, midpoint\_frequency)\*\*2 + wave\_noise(room\_power, fwhm)\*\*2)

    print(wave\_noise(room\_power, fwhm))

    print(fwhm)

    print("Total Photon Noise = " + str(total\_photon\_noise) + " W/Hz^0.5")

    #NEP

    dp = hotbar\_room\_power - room\_power

    Response\_Power = dF0\_hotbar/dp

    print("Hotbar power = " + str(dp) + " W")

    print("response " + str(Response\_Power))

    Noise\_Eq\_Power = np.sqrt(spectral\_densities)/Response\_Power

    plt.loglog(frequencies[1:], Noise\_Eq\_Power[1:], color="black")

    plt.ylabel("NEP / WHz$^{-0.5}$")

    plt.xlabel("Frequency / Hz")

    plt.title("NEP vs Frequency for KID " + str(KID\_Number))

    plt.show()

    response\_dict = {}

    with open('response.csv') as csv\_file:

        csv\_reader = csv.reader(csv\_file, delimiter=',')

        line\_count = 0

        for row in csv\_reader:

            if line\_count == 0:

                line\_count += 1

            else:

                temp = row

                if len(temp) == 0:

                    k\_NUMBER = int(line\_count)

                    response = 0

                    response\_dict[k\_NUMBER] = response

                else:

                    k\_NUMBER = int(temp[0])

                    response = float(temp[1])

                    response\_dict[k\_NUMBER] = response

                line\_count += 1

    for i in range(1, data.numkids):

        plt.plot(i, response\_dict[i], marker='x', color='black', markersize=5)

    plt.title("Responsivities of all KIDs")

    plt.ylabel("Responsivity / Hz W^-1")

    plt.xlabel("KID Number")

    plt.ticklabel\_format(useOffset=False)

    plt.show()

    plt.figure()

    two\_hundred\_Hertz\_Index = np.argmax(frequencies > 2000)

    for i in range(1, data.numkids):

        if response\_dict[i] != 0:

            Noise\_Eq\_Power = np.sqrt(spectral\_densities)/response\_dict[i]

            plt.loglog(frequencies, Noise\_Eq\_Power, label="KID " + str(i))

        else:

            Noise\_Eq\_Power = np.zeros(len(frequencies))

            plt.loglog(frequencies, Noise\_Eq\_Power, label="KID " + str(i))

    plt.title("NEP vs frequency")

    plt.ylabel("NEP")

    plt.xlabel("Frequency / Hz")

    plt.show()

    print("last = " + str(frequencies[-1]))

    all\_kid\_NEP = []

    for i in range(1, data.numkids):

        if response\_dict[i] != 0:

            Noise\_Eq\_Power = np.sqrt(spectral\_densities)/response\_dict[i]

            ave\_NEP = np.average(Noise\_Eq\_Power[two\_hundred\_Hertz\_Index:])

            all\_kid\_NEP.append(ave\_NEP)

        else:

            all\_kid\_NEP.append(0)

    fig, ax = plt.subplots()

    N, bins, patches = ax.hist(all\_kid\_NEP, bins=20 ,edgecolor='white', linewidth=1)

    patches[0].set\_facecolor('r')

    for i in range(1, len(patches)):

        patches[i].set\_facecolor('b')

    bins\_size = bins[1] - bins[0]

    plt.figtext(0.5, 0.01, f"Bin Size = ${bins\_size:.2E}"  , ha="center", fontsize=18, bbox={"facecolor":"orange", "alpha":0.5, "pad":5})

    plt.title("Histogram of KID NEP")

    plt.ylabel("Number of KIDs")

    plt.xlabel("NEP / WHz$^{-0.5}$")

    labels = N

    custom\_lines = [Line2D([0], [0], color="r", lw=4),

                    Line2D([0], [0], color='b', lw=4)]

    # Make some labels.

    rects = ax.patches

    for rect, label in zip(rects, labels):

        height = rect.get\_height()

        if label != 0:

            ax.text(rect.get\_x() + rect.get\_width() / 2, height+0.01, label,

                    ha='center', va='bottom')

    ax.legend(custom\_lines, ["KID with Zero NEP / Unresolvable Response", "KID with Non-zero NEP"])

    plt.xticks(bins[::2])

    # height = 25

    # mu = 0.5e-14

    # sigma = 1

    # c = 0

    # p\_guess = [height, mu, sigma, c]

    print(N)

    # #Gaussian curve

    popt\_main, \_ = curve\_fit(Gaussian, bins[1:10], N[1:10], maxfev=20000)

    x = np.linspace(bins[1], bins[10], num=200)

    y = Gaussian(x, \*popt\_main)

    plt.plot(x,y)

    # response\_main = popt\_main[0]/dT

    plt.show()

def get\_power(planck, transmission\_frequency, transmission, wavelength, dnu, T, BFF=1):

    blackbody\_intensity = planck(transmission\_frequency,transmission, T)

    points = len(blackbody\_intensity)

    power = ((sum(blackbody\_intensity))/points)\*(dnu)

    power = (1/BFF)\*power\*wavelength\*\*2

    return power

#define normalized 2D gaussian

def gaus2d(x, y, mean\_x, mean\_y, sigma\_x, sigma\_y, amplitude):

    first\_frac = ((x-mean\_x)\*\*2)/(2\*sigma\_x\*2)

    second\_frac = ((y-mean\_y)\*\*2)/(2\*sigma\_y\*2)

    return amplitude\*np.exp(-(first\_frac+second\_frac))

def calculate\_Beam\_Filling\_Factor(length=100, width=20, FWHM=20, points=5000):

    #Define x and y

    x = np.linspace(0, length, points)

    y = np.linspace(0, length, points)

    x, y = np.meshgrid(x, y) # get 2D variables instead of 1D

    #mean: square box of (longest length)/2

    mean = [length/2, length/2]

    #sigma = FWHM/(sqrt(8log2))

    divisor = np.sqrt(8\*np.log(2))

    sigma = [FWHM/divisor, FWHM/divisor]

    z = gaus2d(x, y, mean[0], mean[1], sigma[0], sigma[1], 1)

    fig = plt.figure()

    ax = fig.add\_subplot(111, projection='3d')

    ax.plot\_surface(x, y, z, cmap="coolwarm", linewidth=0)

    ax.set\_xlabel('x / mm')

    ax.set\_ylabel('y / mm')

    ax.set\_zlabel('Intensity')

    m = cm.ScalarMappable(cmap="coolwarm")

    cbar = plt.colorbar(m)

    cbar.ax.set\_title('Intensity')

    plt.show()

    fig = plt.figure()

    ax = fig.add\_subplot(111, projection='3d')

    ax.plot\_surface(x, y, z, cmap="coolwarm", linewidth=0)

    ax.set\_zticklabels([])

    ax.set\_xlabel('x / mm')

    ax.set\_ylabel('y / mm')

    ax.grid(False)

    m = cm.ScalarMappable(cmap="coolwarm")

    cbar = plt.colorbar(m)

    cbar.ax.set\_title('Intensity')

    plt.show()

    plt.figure()

    #Integral = sum over all points

    Gauss\_integral = np.sum(z)

    #Bar dimensions centered at length/2

    sub = width/2

    first\_cuttoff = int(((mean[0]-sub)/length)\*points)

    second\_cuttoff = int(((mean[0]+sub)/length)\*points)

    #Set outside hotbar dimensions = 0

    z[0:first\_cuttoff,:] = 0

    z[second\_cuttoff:,:] = 0

    #Integrate over all points

    hotbar\_integral = np.sum(z)

    #Calculate BFF

    Beam\_Filling\_Factor = hotbar\_integral/Gauss\_integral

    return Beam\_Filling\_Factor

def shot\_noise(power, nu):

    shot = np.sqrt(2\*power\*const.h\*nu)

    return shot

def wave\_noise(power, dnu):

    wave = power/(np.sqrt(2\*dnu))

    return wave

def planck(frequency,transmission, T=293):

    a = (2\*const.h\*frequency\*\*3)/const.c\*\*2

    b = 1/(np.exp((const.h\*frequency)/(const.Boltzmann\*T))-1)

    #intensity = a\*b\*transmission\*wavelength\*\*2

    intensity = a\*b\*transmission

    return intensity

def transmission\_fwhm(x, y):

    #create a spline of x and freq-np.max(blue)/2 to find fwhm

    spline = UnivariateSpline(x, y-np.max(y)/2, s=0)

    r1, r2 = spline.roots() #find the roots

    fwhm = abs(r2 - r1)

    r1\_r2\_indices = np.where(np.logical\_and(x>=r1, x<=r2))

    midpoint = fwhm/2 + r1

    plt.plot(x/1e9, y, marker = ".", color = "black", label = "data", markersize = 1)

    plt.axvline(x=midpoint/1e9, label="Midpoint", color = "blue", linestyle="--")

    plt.axvspan(r1/1e9, r2/1e9, facecolor='g', alpha=0.5, label="fwhm")

    plt.xlabel("Frequency / GHz")

    plt.ylabel("Transmission Fraction")

    plt.legend(loc='center left', bbox\_to\_anchor=(1, 0.5), fancybox=True)

    plt.show()

    r1 = x[0]

    r2 = x[-1]

    return fwhm, midpoint, r1, r2

#navigate to which file. Default analysis is ch1 for hot bar data

def get\_user\_input(data):

    #Get Freq

    data\_folder\_name = input("Data file: ")

    local\_file = os.getcwd()

    #Uncomment for use:

    #data\_folder\_name = input("Data file: ")

    data = fsab.fsab\_dirfile(local\_file + "\\"  + data\_folder\_name)

    #Plotting Sweep

    while True:

        Input = input("Know the range? (y/n):")

        if Input.lower() == "y":

            break

        KID\_Number = int(input("KID Number?: "))

        #Plot dF0 v Time

        dF0, time = Get\_dF0(data, KID\_Number)

        plt.plot(time, dF0)

        plt.rcParams["figure.dpi"] = 400

        plt.ticklabel\_format(useOffset=False)

        plt.xlabel("time / s")

        plt.ylabel("dF0 / Hz")

        plt.title("Time evolution of dF0 for KID number " + str(KID\_Number))

        plt.show()

        plt.figure()

        Input = input("Again? (y/n):")

        if Input.lower() == "n":

            break

    #Get dF0 and Time

    KID\_Number = int(input("KID Number?: "))

    lower\_range = float(input("Lower bound of time peak range?:"))

    upper\_range = float(input("Upper bound of time peak range?:"))

    return KID\_Number, upper\_range, lower\_range

def Gaussian(x, height, mu, sigma, c):

  f = c + height\*np.exp((-(x-mu)\*\*2)/(2\*(sigma)\*\*2))

  return f

def Get\_IQ\_Sweep(data, KID\_Number):

    IQ\_data = data.sweep[KID\_Number]["z"]

    sweep = data.sweep[KID\_Number]["f"]

    tone\_freq = data.sweep[KID\_Number]["tone\_freq"]

    I = IQ\_data.real

    Q = IQ\_data.imag

    return I, Q, sweep, tone\_freq

def Get\_IQ\_Time(data, KID\_Number):

    t = (data.start\_time - data.stop\_time)

    IQ\_data = data.get\_iq\_data(KID\_Number)

    I = IQ\_data.real

    Q = IQ\_data.imag

    time = np.linspace(0,t, len(I))

    return I, Q, time

def Get\_dF0(data, KID\_Number):

    #Initialize data

    I\_Sweep, Q\_Sweep, sweep, tone\_frequency = Get\_IQ\_Sweep(data, KID\_Number)

    I\_Time, Q\_Time, time = Get\_IQ\_Time(data, KID\_Number)

    #Get didf and dqdf

    step = sweep[1] - sweep[0]

    I\_Base, Q\_Base = 0, 0

    for i in range(0, len(sweep)):

        if sweep[i] == tone\_frequency:

            I\_Base = I\_Sweep[i]

            Q\_Base = Q\_Sweep[i]

            didf = (I\_Sweep[i+1] - I\_Sweep[i-1])/(2\*step)

            dqdf = (Q\_Sweep[i+1] - Q\_Sweep[i-1])/(2\*step)

    #Get di and df

    di = I\_Time - I\_Base

    dq = Q\_Time - Q\_Base

    #Magic Formula

    dF0 = (di\*didf + dq\*dqdf)/(didf\*\*2 + dqdf\*\*2)

    #Return

    return dF0, time

def fourier\_transform(data, KID\_Number):

    #points

    dF0, time = Get\_dF0(data, KID\_Number)

    plt.plot(time,dF0, color="black", linewidth=3)

    plt.xlabel("Time / s")

    plt.ylabel("dF0 / Hz")

    plt.show()

    plt.figure()

    points = len(time)

    MU = time[-1] - time[0]

    FS = points/MU

    f, Pxx\_den = signal.periodogram(dF0, FS)

    return f, Pxx\_den

if \_\_name\_\_ == "\_\_main\_\_":

    main()