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

```python
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")
```

```python
        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]
```

```python
            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()
```