Week 6: 10/11/2021 – Wednesday

1. **Outline of Meeting**

The meeting was held on Zoom. Following from last week's topic, the meeting was focused on characterizing a KID. Building from last week's theory, this week extends for characterizing the measurement of a KID. By using the scattering parameters equations for the KID, by determining the ABCD values which are related to the transmission line, the S21 value can be found for the detector. This week's task was focused on using a Python function given by my supervisor and given parameters to model a KID for a certain temperature range. The result of which is a plot of cascading frequencies each curve corresponding to a detector.

2. **Specification of Task**
   i) Find Lint using Mattis Bardeen Approximations for Aluminium (use values given last time) at a temperature of 0.2K. Multiply this by "Squares" to get total Lint.
   ii) Find the IDC value for the lowest temperature and make this fixed. Found from F0 and (Lint+Lgeo) and C_couple.

$$\omega_0 = \frac{1}{\sqrt{(L_{int}+L_{Geo})(C_{IDC}+C_{Couple})}}$$

   iii) Run simulations code below to obtain S21 and plot this as a function of frequency
   iv) Define a temperature step (say 0.02K) and calculate a new Lint using Mattis Bardeen Approximations at this slightly higher temperature.
   v) Simulate again changing only the value of Lint (IDC, L_geo and C_couple remain fixed)
   vi) Repeat for all temperatures up to 0.35K

3. **Outline of Theory and Task Methodology**

Last week discussed about the scattering parameters of the KID. This week, we used the Mattis Bardeen Approximations to determine the population change of the electrons and apply this to L_{int} for the change in internal inductance for a range of temperatures. Following this, the KID can be modelled for a temperature change, corresponding to a detection.

Detailed steps:

i) First, we used the Mattis-Bardeen Approximation for a range of temperatures (from 0.2 to 0.35K with step of 0.02K) and constants defined. Equations:

$$\frac{\sigma_1}{\sigma_n} = \frac{2\Delta(\mathrm{T})}{\hbar\omega}\exp(\left(-\frac{\Delta(0)}{\mathrm{k_B T}}\right)\mathrm{K}_0(\frac{\hbar\omega}{2\mathrm{k_B T}})[2\sinh(\frac{\hbar\omega}{2\mathrm{k_B T}})]$$

$$\frac{\sigma_2}{\sigma_n} = \frac{\pi\Delta(\mathrm{T})}{\hbar\omega}[1-2\exp(\left(-\frac{\Delta(0)}{\mathrm{k_B T}}\right)\exp\left(\frac{-\hbar\omega}{2\mathrm{k_B T}}\right)\mathrm{I}_0(\frac{\hbar\omega}{2\mathrm{k_B T}})]$$

ii) From this, we can calculate the change in σ_2 and σ_1 for a change in temperature.

iii) Following this, we can substitute the values of σ_2 into the LPD expression:

$$\sigma_s = -j\frac{n_s e^2}{\omega m}$$

We take the magnitude of this, so the imaginary number can be ignored. Using this, found values for Cooper-pair popuation n_s.

iv) Then, the value of n_s is substituted into the LPD expression to obtain the LPD:

$$\lambda_L = \sqrt{\frac{m}{\mu_0 n_s e^2}}$$

v) Using the LPD, substitute into the expression for L_{int}:

$$L_{int} = \frac{\mu_0 \lambda}{2} \coth\left(\frac{t}{2\lambda}\right)$$

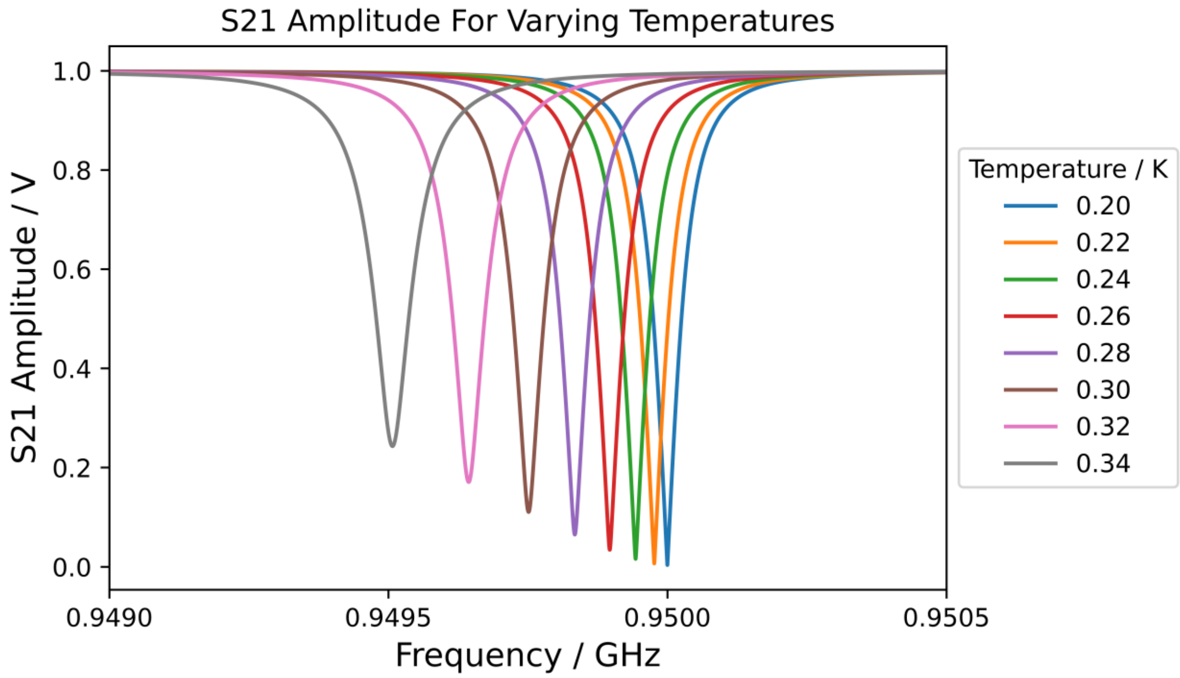
vi) Next, we can find the resistive part R using the following expression:

$$R = L_k \omega \frac{\sigma_1}{\sigma_2}$$

This R accounts for the loss faced by the current when propagating across the quasi-particle as this path is resistive compared to the superconducting n_s.

vii) Finally, plug the R and L_{int} values into the Python function given to obtain S21 and plot the amplitude of S21 against frequency. This figure is generated in the following section.

4. Plot of S21 vs Temperature



This figure illustrates the S21 Amplitude variation with temperature. This is effectively a model of the detector, and this allows us to study and understand the effects of the parameters on its output.

5. Python Function Given:

```
def Capacitive_Res_Sim(F0, C_couple, Z0, L_geo, L_int, Res, Sweep_BW, Sweep_points,
Capacitance):
  j=complex(0,1)
  Cc=C_couple
  F_min=F0-(Sweep_BW/2.0)
```

```
    F_max=F0+(Sweep_BW/2.0)
    Sweep=np.linspace(F_min, F_max, Sweep_points)
    W=Sweep*2.0*pi
    W0=2.0*pi*F0
    L=L_geo+L_int
    C=Capacitance
    Zres= 1.0/((1./((j*W*L)+Res))+(j*W*C))   # Impedance of resonator section
    Zc=1.0/(j*W*Cc)  #impedance of coupler
    ZT=Zres+Zc
    YT=1.0/ZT
    S21 = 2.0/(2.0+(YT*Z0))
    I_raw=S21.real
    Q_raw=S21.imag
    shift=((1.0-min(I_raw))/2.0)+min(I_raw)
    I_cent=I_raw-shift
    Q_cent=Q_raw
    Phase=Atan(abs(Q_cent/I_cent))
    QU=(W0*L)/Res
    QL=(C*2)/(W0*(Cc**2)*Z0)
    S21_Volt=abs(S21)
    I_offset=shift
    return  (Sweep, S21_Volt, Phase, I_raw, Q_raw, I_cent, Q_cent, QU, QL, I_offset)
```

6. Python Code for Task

```
#imports
import numpy as np
import matplotlib.pyplot as plt
import scipy.constants as const
from scipy.special import iv as I0
from scipy.special import kv as K0

#Define Global Variables
L_geo = 55.6e-9
Z0 = 50.0
F0_base = 0.95e9        #At lowest Temp
squares= 27223
c_couple = 1.5e-14
TC = 1.5
Delta_0 = (3.5*const.Boltzmann*TC)/2
sigma_n = 6.0e7          # Normal stae conductvity if superconducting film
Thick = 20e-9            # Thickness of superconducting fil
w = 2 * np.pi * F0_base
me = const.m_e
miu_0 = 4*np.pi*10**-7
pi = np.pi

#Main code
def main():
    #Define temperature range with step 0.01K
    step = 0.02
    temp = np.arange(0.20, 0.35, step)
```

```
    #Find sigma1 and sigma 2 and Lint
    sigma1, sigma2 = find_sigma1_sigma2(sigma_n ,Thick, TC, Delta_0, w, temp)
    Lint = find_Lint_square(Thick, w, sigma2) * squares

    #Find lk
    Lk = find_lk(Thick, w, sigma2)

    #Find Res
    sigma12Ratio = sigma1/sigma2
    Res = Lk*w*sigma12Ratio *squares

    #IDC for Lowest Temp (0.2K)
    Ltot_lowest = Lint[0] + L_geo
    IDC = find_IDC(w, Ltot_lowest, c_couple)

    #Find S21
    Sweep_points = 20000
    BW = 5e6
    I_raw = np.zeros((Sweep_points, len(temp)), dtype="float")
    Q_raw = np.copy(I_raw)
    Phase = np.copy(Q_raw)
    S21_Volt = np.copy(I_raw)
    for i in range(0, len(Lint)):
        Sweep, S21_Volt[:,i], Phase[:,i], I_raw[:,i], Q_raw[:,i],_,_,_,_,_ = Capacitive_Res_Sim(F0_base,
c_couple, Z0, L_geo, Lint[i], Res[i], BW, Sweep_points, IDC)
        plt.plot(Sweep/1e9, S21_Volt[:,i], label=str("{:.2f}".format(temp[i])))

    #Graph labels and title
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fancybox=True, title="Temperature / K")
    plt.xlabel('Frequency / GHz', fontsize=13)

    plt.ylabel('S21 Amplitude / V', fontsize=13);
    plt.title("S21 Amplitude For Varying Temperatures")
    plt.xlim(0.9490, 0.9505)
    plt.locator_params(nbins=6)
    plt.savefig("S21 Plot with Resistance")
    plt.rcParams['figure.dpi'] = 300
#KID Simulating Function
def Capacitive_Res_Sim(F0, C_couple, Z0, L_geo, L_int, Res, Sweep_BW, Sweep_points,
Capacitance):
    """ Help file here"""
    j=complex(0,1)
    Cc=C_couple
    F_min=F0-(Sweep_BW/2.0)
    F_max=F0+(Sweep_BW/2.0)
    Sweep=np.linspace(F_min, F_max, Sweep_points)
    W=Sweep*2.0*pi
    W0=2.0*pi*F0
    L=L_geo+L_int
    C=Capacitance
```

```
    Zres= 1.0/((1./((j*W*L)+Res))+(j*W*C))   # Impedance of resonator section
    Zc=1.0/(j*W*Cc)  #impedance of coupler
    ZT=Zres+Zc
    YT=1.0/ZT
    S21 = 2.0/(2.0+(YT*Z0))
    I_raw=S21.real
    Q_raw=S21.imag
    shift=((1.0-min(I_raw))/2.0)+min(I_raw)
    I_cent=I_raw-shift
    Q_cent=Q_raw
    Phase=Atan(abs(Q_cent/I_cent))
    QU=(W0*L)/Res
    QL=(C*2)/(W0*(Cc**2)*Z0)
    S21_Volt=abs(S21)
    I_offset=shift
    return  (Sweep, S21_Volt, Phase, I_raw, Q_raw, I_cent, Q_cent, QU, QL, I_offset)

#Function to find sigma1 and sigma2
def find_sigma1_sigma2(sigma_n ,Thick, TC, Delta_0, w, T):
    #An interpolation formula for delta_T
    delta_T = Delta_0*np.tanh(1.74*np.sqrt((TC/T)-1))

    #Define constants to simplify eqn
    multiplying_constant = delta_T/(const.hbar * w)
    e_const_1 = - Delta_0/(const.Boltzmann*T)
    e_const_2 = (const.hbar*w)/(2*const.Boltzmann*T)

    #Parts of the sigma1 Ratio
    A = 2*multiplying_constant
    B = np.exp(e_const_1)
    C = K0(0, e_const_2)
    D = 2*(np.sinh(e_const_2))

    #Find Sigma 1 and Sigma 2
    sigma1Ratio = A * B * C * D
    sigma2Ratio = np.pi*multiplying_constant*(1 - (2*np.exp(e_const_1)*np.exp(-
e_const_2)*I0(0,e_const_2)))
    sigma2 = sigma2Ratio * sigma_n
    sigma1 = sigma1Ratio * sigma_n
    return sigma1, sigma2

def find_lk(Thick, w, sigma2):
    #Depth
    lower_fraction = miu_0*sigma2*w
    Lambda_T_MB = (1/lower_fraction)**0.5
    fraction = Thick/(2*Lambda_T_MB)

    #Terms for lk
    A = (miu_0*Lambda_T_MB)/4
    B = coth(fraction)
    C = fraction*(csch(fraction))**2
```

```
    #R vs T
    lk = A*(B+C)
    return lk

def find_Lint_square(Thick, w, sigma2):
    #Depth
    lower_fraction = miu_0*sigma2*w
    Lambda_T_MB = (1/lower_fraction)**0.5

    #Internal Inductance
    fraction = Thick/(2*Lambda_T_MB)
    L_int = (miu_0*Lambda_T_MB/2)*coth(fraction)
    return L_int

#Define coth and csch
def coth(x):
    return np.cosh(x)/np.sinh(x)

def csch(x):
    return 1/np.sinh(x)

def Atan(x):
    return np.arctan(x)

#Find IDC function
def find_IDC(w0, Ltot, Cc):
    IDC = 1/((w0**2)*Ltot) - Cc
    return IDC

def Magic_Formula(di, dq, didf, dqdf):
    return (di*didf + dq*dqdf)/(didf**2 + dqdf**2)

main()
```