**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.

1. **Specification of Task**
2. 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.
3. Find the IDC value for the lowest temperature and make this fixed. Found from F0 and (Lint+Lgeo) and C\_couple.
4. Run simulations code below to obtain S21 and plot this as a function of frequency
5. Define a temperature step (say 0.02K) and calculate a new Lint using Mattis Bardeen Approximations at this slightly higher temperature.
6. Simulate again changing only the value of Lint (IDC, L\_geo and C\_couple remain fixed)
7. Repeat for all temperatures up to 0.35K
8. **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 Lint 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:

1. 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:
2. From this, we can calculate the change in and for a change in temperature.
3. Following this, we can substitute the values of into the LPD expression:

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

1. Then, the value of is substituted into the LPD expression to obtain the LPD:
2. Using the LPD, substitute into the expression for Lint:
3. Next, we can find the resistive part R using the following expression:

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 ns.

1. Finally, plug the R and Lint 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.
2. **Plot of S21 vs Temperature**

Diagram

Description automatically generated

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.

1. **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)

1. **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()