INDIAN INSTITUTE OF SCIENCE

IISC QUANTUM TECHNOLOGY INITIATIVE

QT211 - BASIC QUANTUM TECHNOLOGY LABORATORY

Experiment 1

Chaitali Shah
(SR No: 01-02-04-10-51-21-1-19786)

Date: November 29, 2023

Aim of the Experiment

1. Time domain reflectometry measurements

    (a) Understanding characteristic impedance of a cable

    (b) Establish wave nature of EM signals

    (c) To understand short, open and matched load boundary conditions

Oscilloscope

1. Given is the schematic for the measurement setup using the oscilloscope. The pulse generator has an internal impedance of 50 Ω and Oscilloscope impedance has been set to MΩ. Usually, modern oscilloscopes have two options for the input signal ports. The input impedance can be set either high impedance 1 MΩ or 50 Ω.

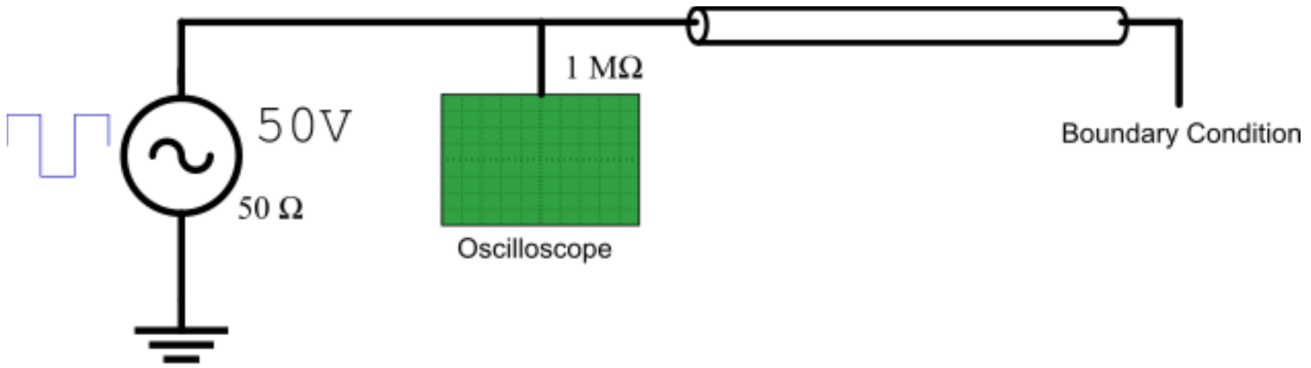


Figure 1: Schematic for Reflectometry Measurement

First, we are going to look at the steady state response. Let us replace the oscilloscope with a Digital Multimeter (DMM) and the Pulse generator with a DC voltage source. Calculate the voltage measured by the DMM when the cable is terminated with an open, short or 50 Ω termination.

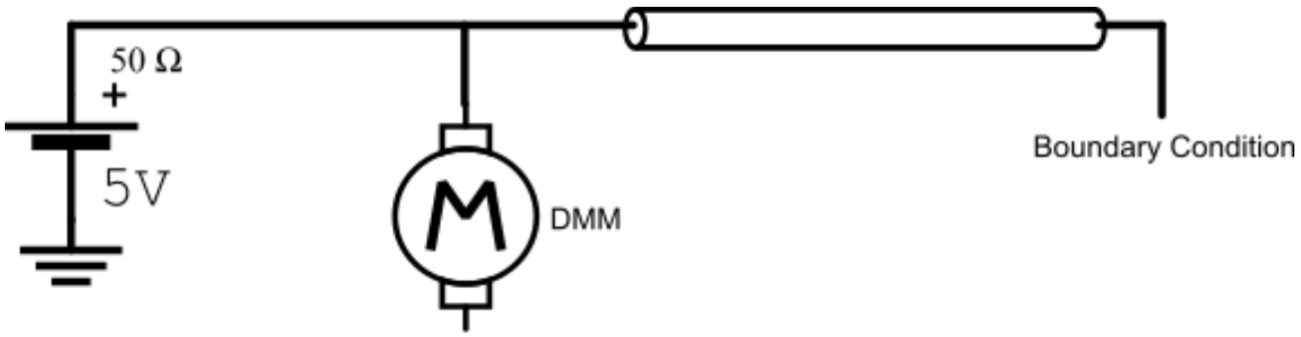


Figure 2: Schematic for Reflectometry Measurement

(a) The cable is terminated with an open termination :
By DC analysis, the open-circuited line presents an open circuit at its terminals. This results in a terminal voltage that is equal to the source voltage. For steady state in an open circuit,

$$V_{Measured} = V_{DC} = 5V$$

(b) The cable is terminated with a short termination:
With DC circuit analysis, the shorted line has zero electrical length at DC and thus appears as a short at its input. At steady state this leads to a terminal voltage of zero.

$$V_{Measured} = V_{DC} = 0V$$

(c) The cable is terminated with a 50 Ω termination:
For a DC source, this network at steady state would appear as a voltage divider consisting of equal source and input impedances. So the voltage measured will be,

$$V_{Measured} = (5)\frac{50}{50+50} = 2.5V$$

2. Using the attached oscilloscope data set:

   (a) Plot graphs for the time domain measurement in Python.

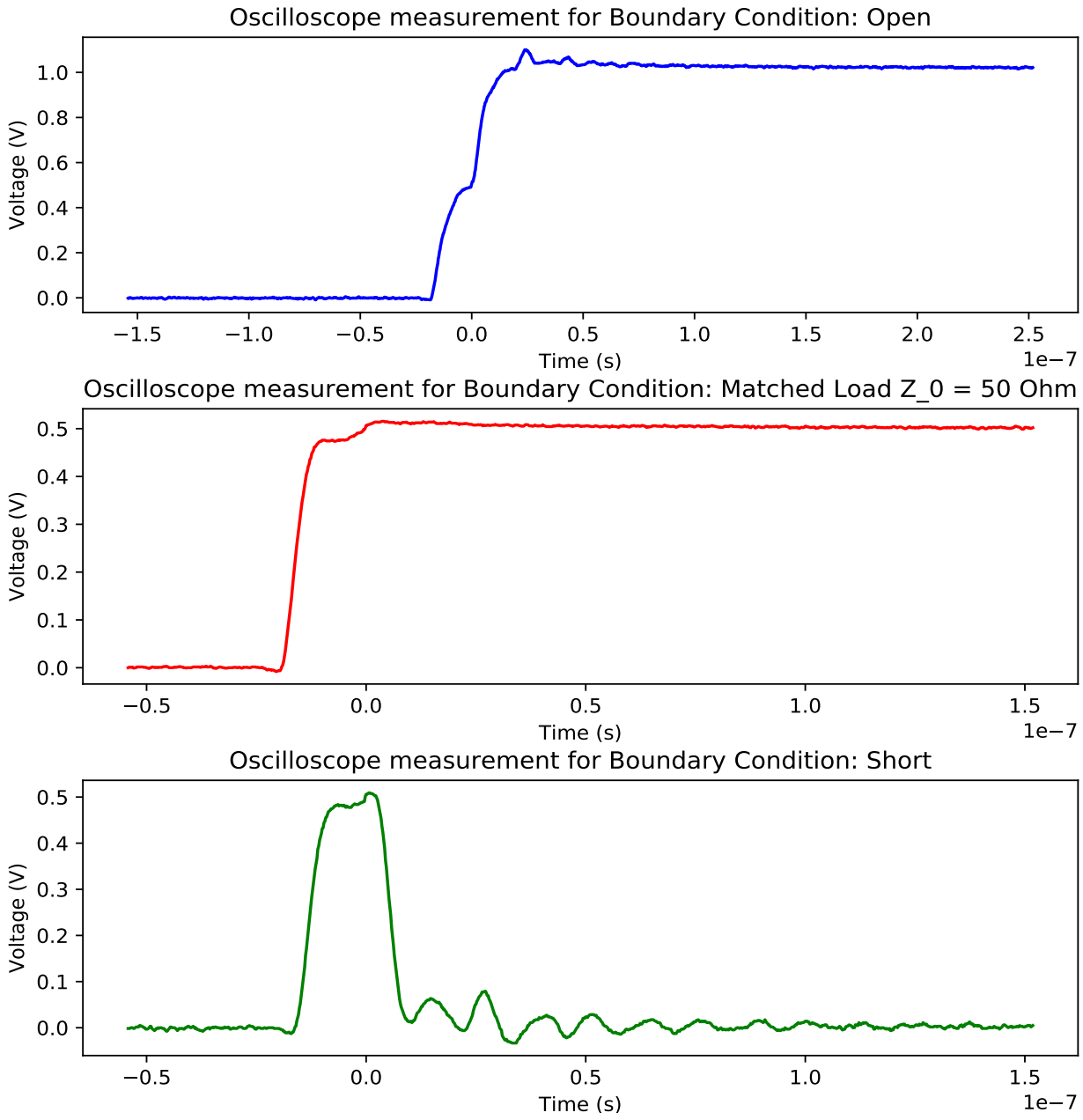


Figure 3: Time Domain Measurement

(b) Compute the group velocity (Vg) of the signal in the cable. From the shared experimental data, estimate error in the calculation of Vg. Length of the cable used to record the data = 3 ft.

i. When the cable is terminated with an open termination the impedance of the line will appear to be equal to the characteristic impedance initially for $t < \frac{l}{v_g}$, and initially the amplitude of the voltage is $\frac{V_0}{2} = 0.5V$. The open-circuit load has a reflection coefficient of

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} = \lim_{Z_L \to \infty} \frac{Z_L - 50}{Z_L + 50} = 1$$

which reflects the incident waveform with the same polarity toward the source. The amplitudes of the forward and reverse pulses add to create a wave with an amplitude of $V = V_0 = 1V$, as shown in Figure 3, for the open boundary. At $t = \frac{2l}{v_g}$ the return pulse reaches the source, but it is not re-reflected since the source is matched to the line. Hence, the group velocity is given by

$$v_g = \frac{2l}{t}$$

From the plots, $t = 8ns$ and the length of the cable is, l = 3 ft = 0.9144 m. Hence,

$$v_g = \frac{2l}{t} = \frac{2 \times 0.9144}{8 \times 10^{-9}} = 2.286 \times 10^8 s$$

ii. When cable is terminated with a short termination, again the amplitude of the voltage is $\frac{V_0}{2} = 0.5V$ initially. The short-circuit load has a reflection coefficient of

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} = \frac{0 - 50}{0 + 50} = -1$$

and the reflected wave subtracts with the amplitude of the incident wave to give $V = 0V$, as shown in Figure 3, for the short termination. At $t = \frac{2l}{v_g}$ the return pulse reaches the source, hence the group velocity is given by

$$v_g = \frac{2l}{t}$$

From the plots, $t = 7.6ns$ and the length of the cable is, l = 3 ft = 0.9144 m. Hence,

$$v_g = \frac{2l}{t} = \frac{2 \times 0.9144}{7.6 \times 10^{-9}} = 2.406 \times 10^8 m/s$$

iii. When the cable is terminated with a 50 Ω termination, the pulse reaches the load at time $t = \frac{l}{v_g}$. Since the load is matched to the line, there is no reflection of the pulse from the load and the the voltage is $V_0 = 0.5V$ throughout.

The group velocity for this cable that has solid PTFE as the dieletric is given by

$$v_g = c/\sqrt{\epsilon_r}$$

where $\epsilon_r = 2.02$.

$$v_g = \frac{3 \times 10^8}{\sqrt{2.02}} = 2.1037 \times 10^8 m/s$$

Tolerance	Time delay	Group Velocity	Error percentage
Open termination			
5 %	8 ns	2.286 $\times 10^8$	9.22 %
10 %	10.8 ns	1.685 $\times 10^8$	19.8 %
15 %	13.2 ns	1.378 $\times 10^8$	34.4 %
Short termination			
3 %	7.6 ns	2.406 $\times 10^8$	14.38 %
5 %	12 ns	1.524 $\times 10^8$	27.55 %
8 %	14 ns	1.306 $\times 10^8$	37.90 %

Table 1: Group velocity and error percentages for various tolerance levels for the Open and Short termination

3. What could be alternate approaches if one wants to reduce the error in V_g estimates?

   The signal generator cannot generate an ideal square wave and hence the transitions in the wave forms are not clearly visible. There is a finite rise time for the pulse generated by the signal generator and to get better estimates of group velocity it is important to be able to distinguish the rise-time transients from the actual variations in the plot.

   It can be seen in the table above that for different tolerance levels about the stable region, that is used for the measurement of time delay, the values of group velocity obtaind are significantly different. In order to reduce the error in V_g estimates it is important to have a sufficiently good resolution for the oscilloscope along with accurate signal generators.

   Also, in the case of the matched load, there is a small deformity in the waveform before the voltage to a value close to 0.5 V. This shows that there is a slight reflection in the practical setup, possibly due to the load impedance being slightly higher than 50 Ω. Similary, the short termination is also not ideal and might have some small impedance than can add error to the v_g estimate. There can be improvement in resolution, accuracy of measuring instruments and the use of engineering judgement to obtain reasonable estimates.

Theory Question

1. (a) Evaluate scattering parameters for 3 dB attenuator circuit.

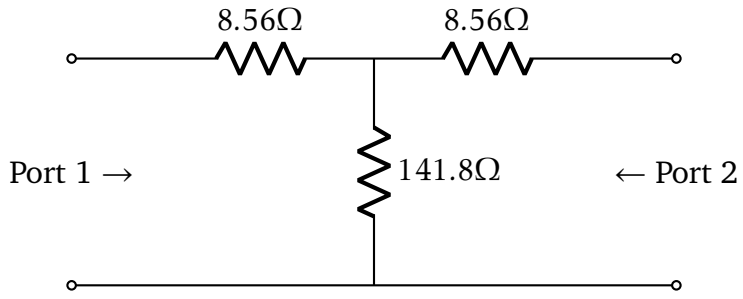


Figure 4: Matched 3 dB attenuator with a 50 Ω characteristic impedance

A specific element of the scattering matrix can be determined as

$$S_{ij} = \left.\frac{V_i^-}{V_j^+}\right|_{V_k^+=0 \text{ for } k\neq 0} \tag{1}$$

S_{11} can be found as the reflection coefficient seen at port 1 when port 2 is terminated in a matched load ($Z_0 = 50\ \Omega$):

$$S_{11} = \left.\frac{V_1^-}{V_1^+}\right|_{V_2^+=0} = \Gamma^{(1)}\big|_{V_2^+=0} = \left.\frac{Z_{in}^{(1)} - Z_0}{Z_{in}^{(1)} + Z_0}\right|_{Z_0 \text{ on port 2}}$$

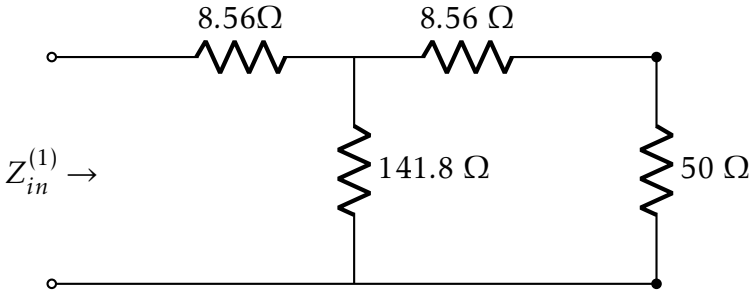


Figure 5: Calculation of S_{11}

$$Z_{in}^{(1)} = 8.56 + \frac{141.8(8.56+50)}{(141.8+8.56+50)} = 50\ \Omega$$

So $S_{11} = 0$.
Because of the symmetry of the circuit, $S_{22} = 0$.
S_{21} is found by applying an incident wave at port 1, V_1^+ , and measuring the outcoming wave at port 2, V_2^-. This is equivalent to the transmission coefficient from port 1 to port 2:

$$S_{21} = \left.\frac{V_2^-}{V_1^+}\right|_{V_2^+=0}$$

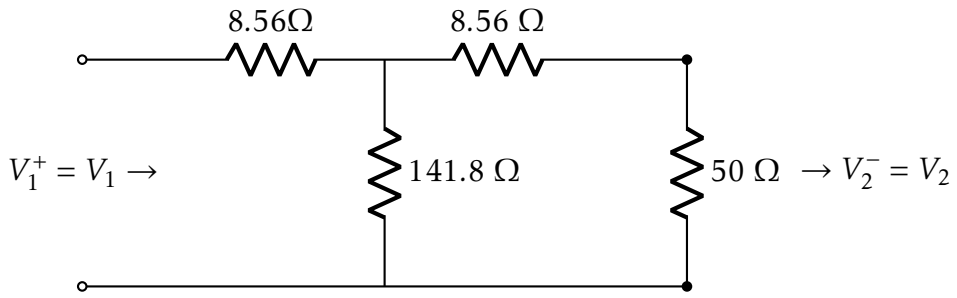


Figure 6: Calculation of S_{21}

We have found that $S_{11} = S_{22} = 0$, so we know that $V_1^- = 0$ when port 2 is terminated in $Z_0 = 50\ \Omega$, and that $V_2^+ = 0$. In this case we have that $V_1^+ = V_1$ and $V_2^- = V_2$. By applying a voltage V_1 at port 1 and using voltage division twice we find $V_2^- = V_2$ as the voltage across the 50 Ω load resistor at port 2:

$$V_2^- = V_2 = V_1 \frac{(41.11)}{(41.11 + 8.56)} \frac{(50)}{(50 + 8.56)} = 0.707 V_1$$

where

$$41.44\ \Omega = \frac{141.8(58.56)}{(141.8 + 58.56)}$$

is the resistance of the parallel combination of the 50 Ω load and the 8.56 Ω resistor with the 141.8 Ω resistor.
Thus, $S_{12} = S_{21} = 0.707$.
The scattering matrix is given as:

$$[S] = \begin{bmatrix} 0 & 0.707 \\ 0.707 & 0 \end{bmatrix}$$

(b) Convert the scattering matrix into impedance matrix.
It is known that

$$[Z] = Z_0([U] + [S])([U] - [S])^{-1}$$

where

$$[U] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and

$$[S] = \begin{bmatrix} 0 & 0.707 \\ 0.707 & 0 \end{bmatrix}$$

and

$$Z_0 = 50\Omega$$

$$[Z] = 50\begin{bmatrix} 1 & 0.707 \\ 0.707 & 1 \end{bmatrix}\begin{bmatrix} 1 & -0.707 \\ -0.707 & 1 \end{bmatrix}^{-1} \Omega$$

$$[Z] = 50\begin{bmatrix} 1 & 0.707 \\ 0.707 & 1 \end{bmatrix}\begin{bmatrix} 1.9994 & 1.4136 \\ 1.4136 & 1.9994 \end{bmatrix} \Omega$$

$$[Z] = 50\begin{bmatrix} 2.9988 & 2.8271 \\ 2.8271 & 2.9988 \end{bmatrix} \Omega$$

$$[Z] = \begin{bmatrix} 150.36 & 141.8 \\ 141.8 & 150.36 \end{bmatrix} \Omega$$

Appendix: Codes

The Python code to plot the time domain measurements for the various boundary conditions is given below.

```
#importing required libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
#adjusting default figure size
plt.rc('figure', figsize=(8, 8))

#The open boundary condition
#Reading the Wfm.csv file
V_open = pd.read_csv('Open.Wfm.csv')
#Defining the horizontal time axis using
#HardwareXStart, HardwareXStop, Resolution
t_open = np.arange( start= -1.542e-007, stop= 2.522e-007,step = 2e-010)
#Plotting the data
plt.figure(1)
plt.subplot(311)
plt.plot(t_open[0:len(V_open)],V_open,'b')
plt.title('Oscilloscope measurement for Boundary Condition: Open')
plt.ylabel('Voltage (V)')
plt.xlabel('Time (s)')

#The matched load Z_0=50 Ohm boundary Condition
#Reading the .csv file
V_match = pd.read_csv('50Ohm.Wfm.csv')
#Defining the horizontal time axis using
#HardwareXStart, HardwareXStop, Resolution
t_match = np.arange( start= -5.42e-008, stop=1.522e-007,step = 2e-010)
#Plotting the data
plt.subplot(312)
plt.plot(t_match[0:len(V_match)],V_match,'r')
plt.title('Oscilloscope measurement for Matched Load Z_0 = 50 Ohm')
plt.ylabel('Voltage (V)')
plt.xlabel('Time (s)')

#The dead-short boundary condition
#Reading the .csv file
V_short = pd.read_csv('Short.Wfm.csv')
#Defining the horizontal time axis using
#HardwareXStart, HardwareXStop, Resolution
t_short = np.arange( start= -5.42e-008, stop=1.522e-007,step = 2e-010)
```

```
#Plotting the data
plt.subplot(313)
plt.plot(t_short[0:len(V_short)],V_short,'g')
plt.title('Oscilloscope measurement for Boundary Condition: Short')
plt.ylabel('Voltage (V)')
plt.xlabel('Time (s)')

#Displaying the plots and adjusting size
plt.subplots_adjust(top=0.95,bottom=0.05, left=0.10, right=0.95, hspace=0.35)
plt.show()
```

INDIAN INSTITUTE OF SCIENCE

IISC QUANTUM TECHNOLOGY INITIATIVE

QT211 - BASIC QUANTUM TECHNOLOGY LABORATORY

Experiment 2

Chaitali Shah
(SR No: 01-02-04-10-51-21-1-19786)

Date: November 29, 2023

Aim of the Experiment

1. Microwave measurements using a Vector Network Analyzer (VNA)

    (a) Estimating dielectric properties of cable

    (b) To understand different scales of representation in VNA

2. Measure the characteristics of a Directional Coupler

Vector Network Analyzer

1. Plot the Phase (Use S_{11}) and explain the oscillations with frequency.

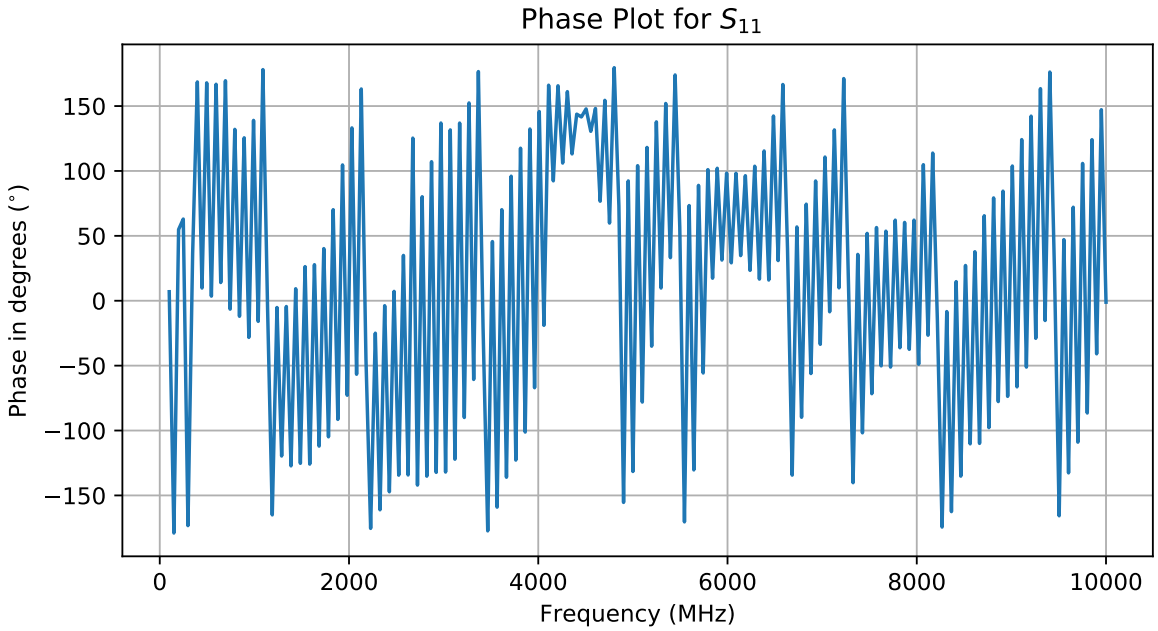


Figure 1: The Phase plot of S_{11}

A element S_{11} of the scattering matrix can be determined as:

$$S_{11} = |S_{11}|e^{i\phi} = \left.\frac{V_1^-}{V_1^+}\right|_{V_2^+=0}$$

For a lossless line, $V(z) = V_1^+ e^{-i\beta z} + V_1^- e^{i\beta z}$.
Due to finite cable length l, the wave travels z=-2l for the measurement of S_{11} and we have,

$$S_{11} = |S_{11}|e^{i\phi} = \frac{|V_1^-|}{|V_1^+|}e^{j\beta 2l}$$

$$\phi = \beta 2l = \omega\sqrt{LC} \times 2l = (2\pi f)2l\sqrt{LC} = (f)4\pi l\sqrt{LC} \tag{1}$$

The plot in Fig. 1 is the plot of phase, ϕ versus the frequency. f. It is seen in the plot that the phase changes linearly, starting from about $-180°$ going to $180°$, followed by an abrupt *winding* back to $-180°$ and resuming the same pattern. The linear dependence is expected as we know from Eq. (1) that the phase is directly proportional to frequency. The winding is due to the modulo $360°$ arithmetic. The minor fluctuations along the plot seem to be due to noise of another frequency.

2. Use the attached data sheet and S_{21} data to estimate dielectric constant of the cable.

   From **[1]** and **[2]**, we have the following equations (Nicholson-Ross-Wier method):
   The reflection coefficient can be expressed as $\Gamma = K \pm \sqrt{k^2 - 1}$ where

$$K = \frac{S_{11}^2(\omega) - S_{21}^2(\omega) + 1}{2S_{11}^2(\omega)}$$

   The + or - in the expression in chosen such that $|\Gamma| \leq 1$.

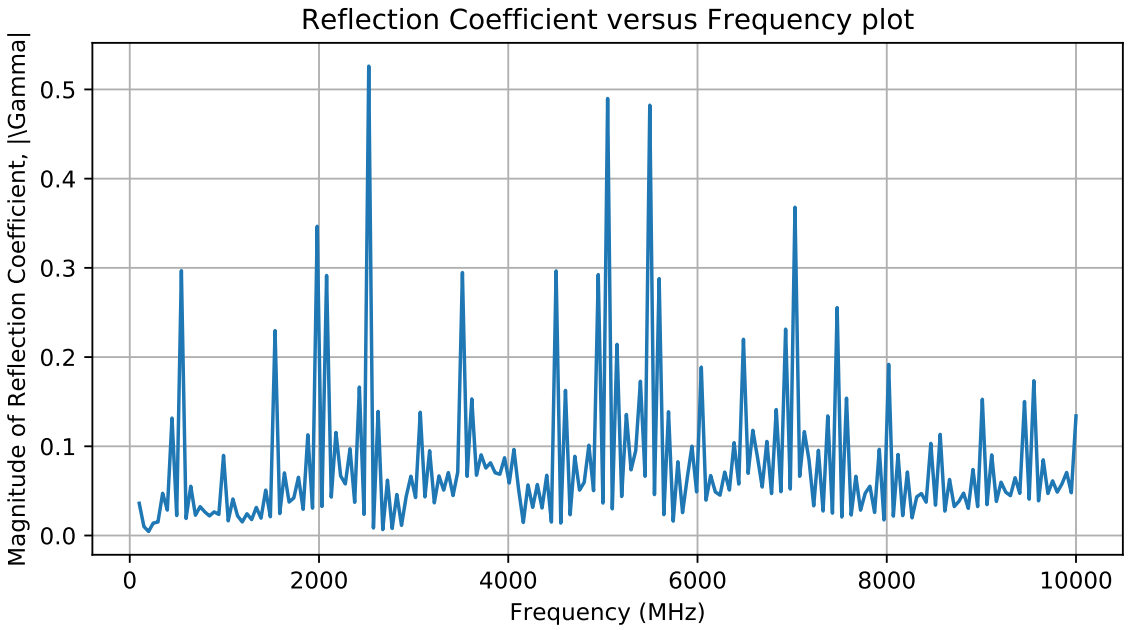


Figure 2: Reflection coefficient as a function of frequency

The transmission coefficient is given by,

$$T = \frac{S_{11}(\omega) + S_{21}(\omega) - \Gamma}{1 - (S_{11}(\omega) + S_{21}(\omega))\Gamma} \tag{2}$$

Defining the auxillary variables x and y as follows,

$$x = \left(\frac{1+\Gamma}{1-\Gamma}\right)^2 \qquad y = -\left(\frac{c}{\omega d}\ln\left(\frac{1}{T}\right)\right)^2$$

where c = speed of light in free space, ω = frequency in radians,
d = sample thickness = $10.67 - 7.92$ mm = 2.75 mm (from datasheet).
We can obtain relative permittivity of the material as,

$$\epsilon_r = \sqrt{\frac{y}{x}} = \epsilon' - i\epsilon''$$

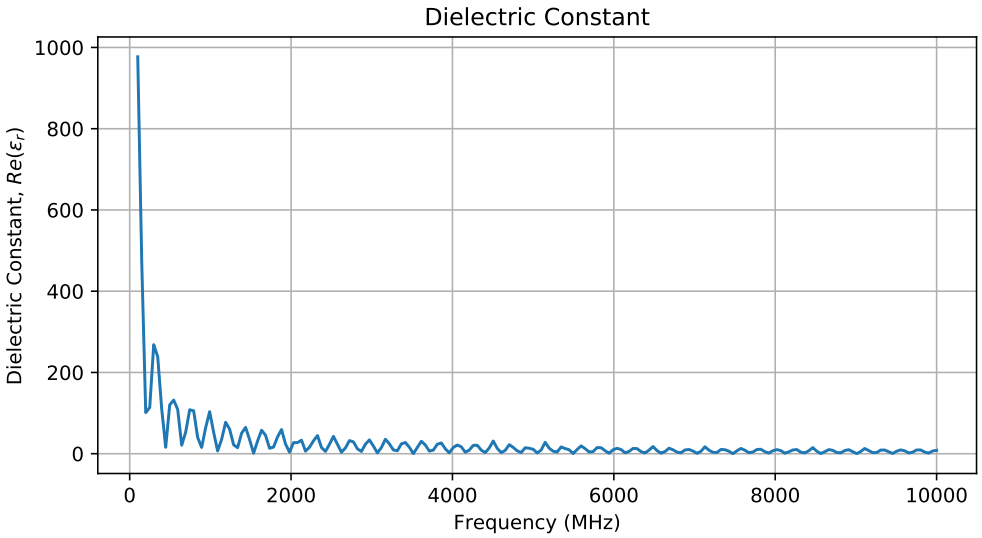


Figure 3: Dielectric Constant as a function of frequency

Fig. 3 is zoomed in to obtain the value of the dielectric constant. It is seen in Fig. 4 that the value oscillates around 2.1.

$$\text{Dielectric constant} = Re(\epsilon_r) \approx 2.1$$

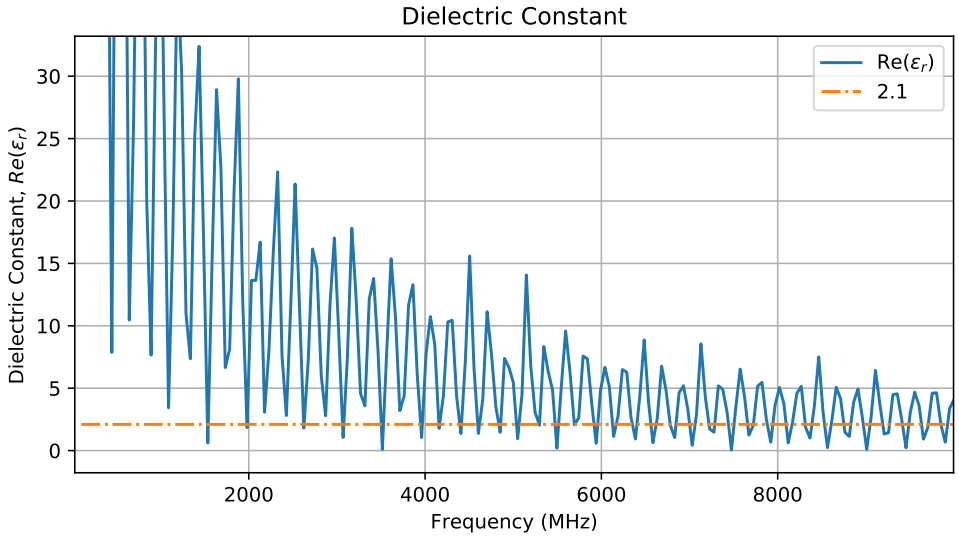


Figure 4: Dielectric Constant = 2.1

3. Given data set is in linear scale. Plot the same in logarithmic scale (dB).

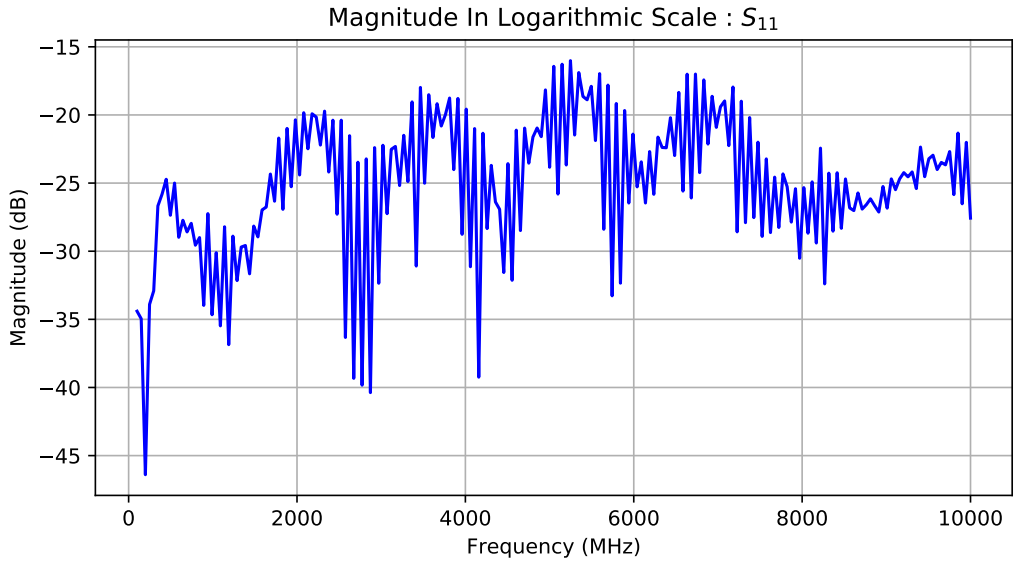


Figure 5: Magnitude Plot of S_{11} in the Logarithmic Scale

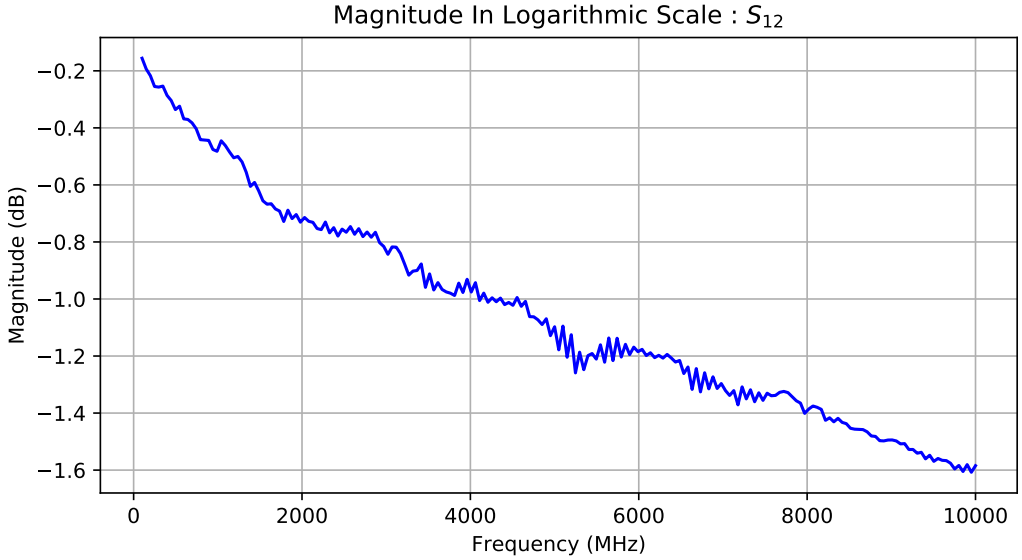


Figure 6: Magnitude Plot of S_{12} in the Logarithmic Scale

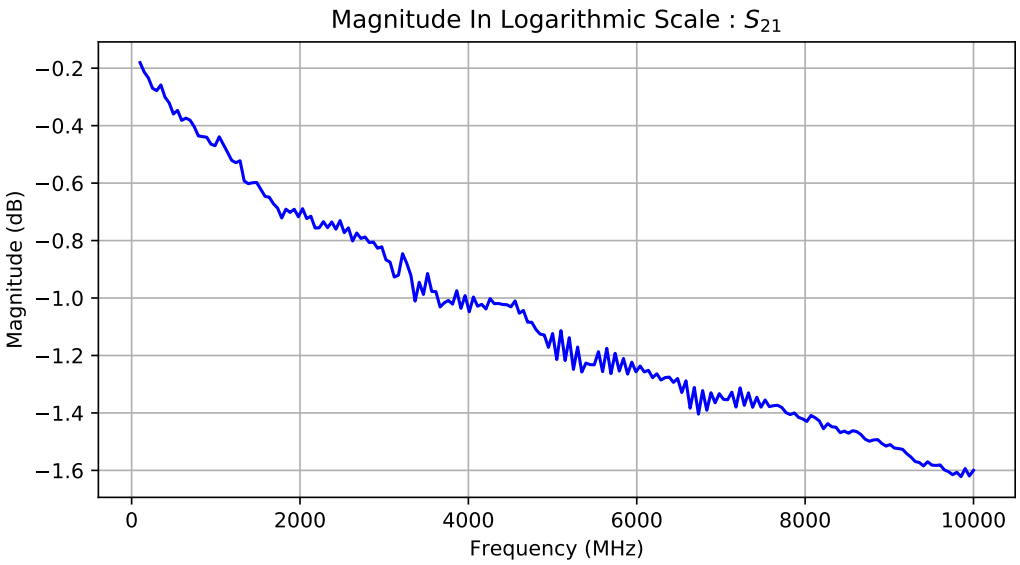


Figure 7: Magnitude Plot of S_{21} in the Logarithmic Scale

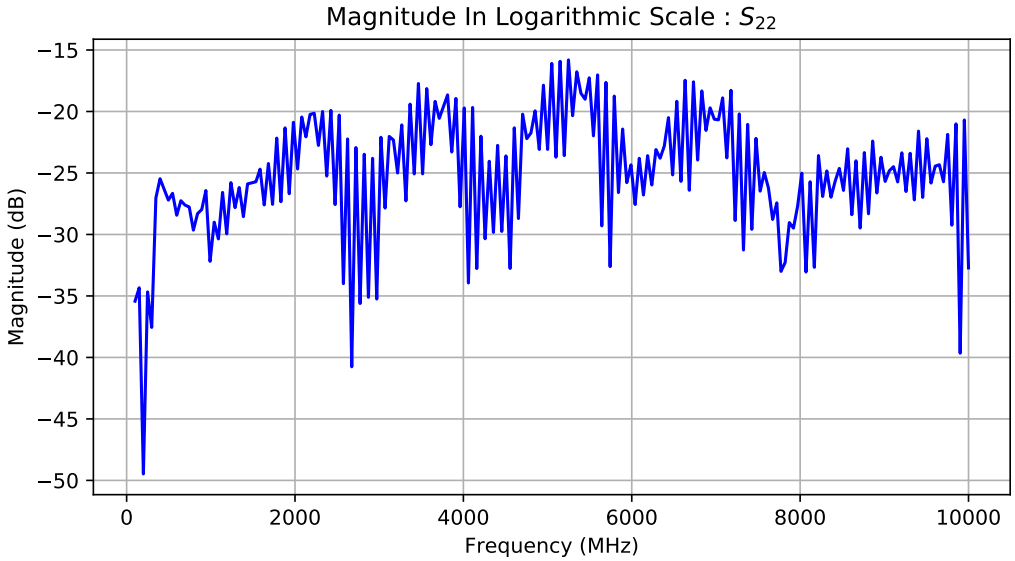


Figure 8: Magnitude Plot of S_{22} in the Logarithmic Scale

The plots shown in Fig. 5,6,7 and 9 were obtained by using the following equation.

$$S_{ij,dB} = 20\log(|S_{ij}|)\ dB$$

4. Assuming only dielectric loss

   (a) Plot the Transmission Coefficient vs frequency graph.
   Using Eq. (2), we obtain the following plot.

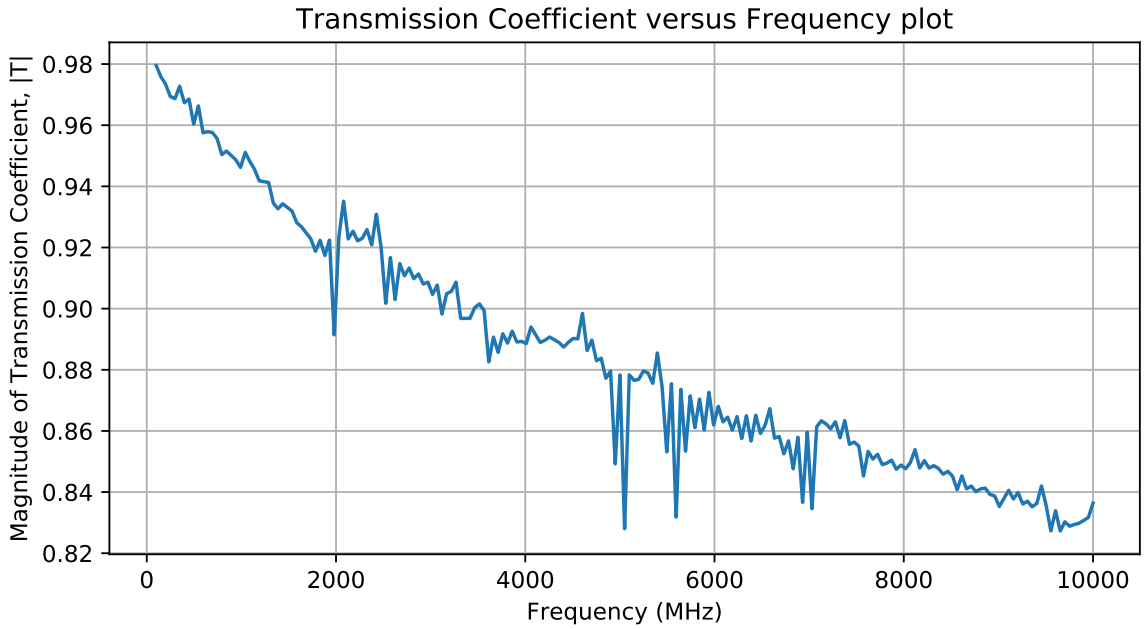


Figure 9: Magnitude Plot of Transmission Coefficient, |T|

   (b) Compute length of the cable.
   In Fig. 5 we can see that the difference between the frequencies of two dips is

$$\Delta f \approx 100MHz$$

$$\text{Length of cable}, l = \frac{c}{2\sqrt{\epsilon_r}\Delta f}$$

$$l = \frac{3 \times 10^8 m/s}{2 \times \sqrt{2.1} \times 0.1 \times 10^9 Hz}$$

$$l \approx 1.035\ m \approx 3.3\ ft$$

(c) Compute the loss tangent of Teflon.
Loss tangent = $\tan(\delta) = \tan(\frac{\epsilon''}{\epsilon'})$
Though not very evident in the plot in Fig. 10, as the value keeps fluctuating about some value very close to zero, it is known that:
Loss tangent = $\tan(\delta) \approx 0.0002$

Figure 10: Loss Tangent $\tan(\delta)$ plot

5. Concept of reference planes: Given the phase values of the S parameters for two different lengths of the cables with the other port kept – OPEN.

   (a) Plot the Phase vs Frequency and explain qualitatively the shift in the graph.

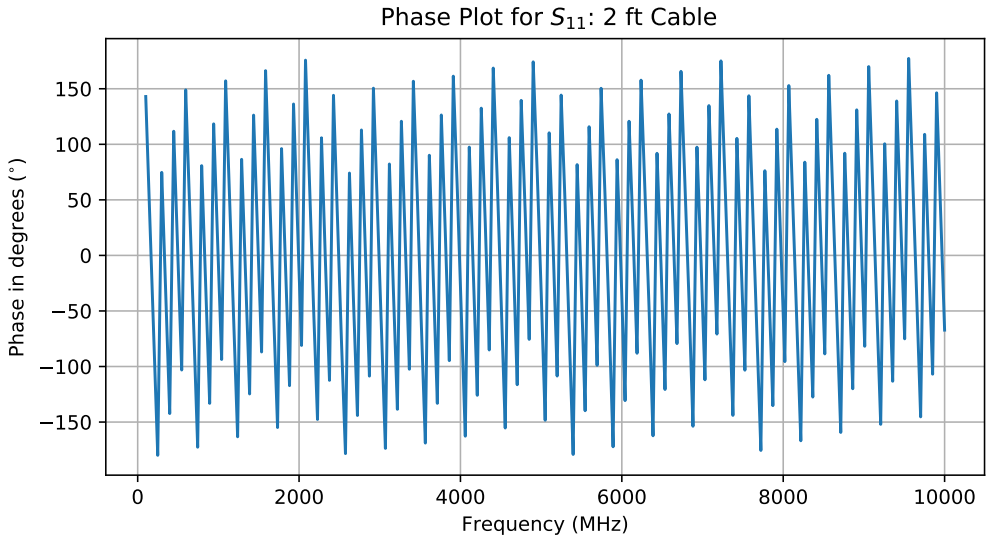


Figure 11: The Phase plot for two different lengths of cables

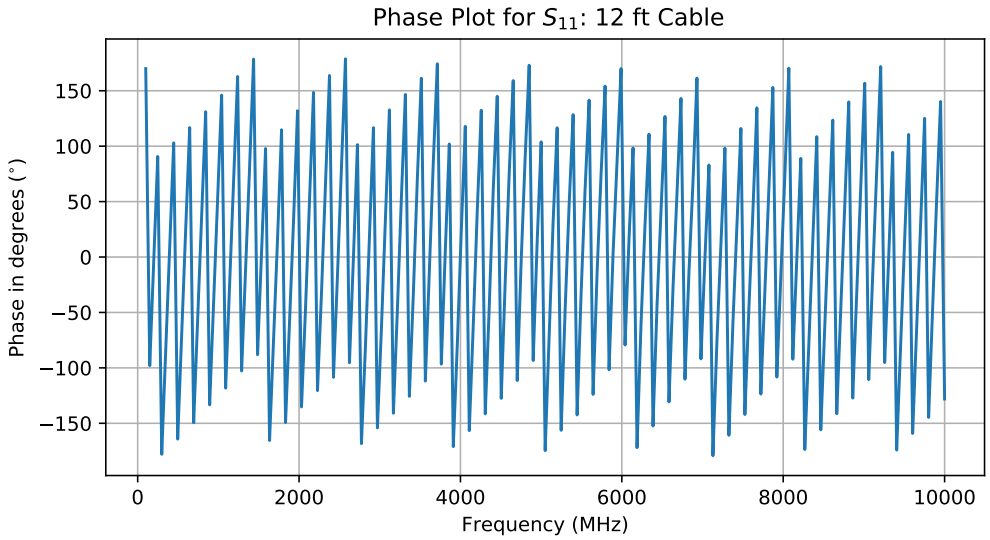


Figure 12: The Phase plot for two different lengths of cables

The phase depends on the length of the cable as seen in Eq. (1). For the greater length of the cable the phase is higher and so the shorter cable as lower values of phase than the longer cable.

(b) Compute the observed phase shift and explain how it is related to the length of the cable.

From the theory of traveling waves on lossless transmission lines we can relate the wave amplitudes of one cable to the other ones as

$$V_2^+ = V_1^+ e^{2j\theta}, \quad V_2^- = V_1^- e^{-2j\theta}$$

where $\theta = \beta l$ is the electrical length of the outward shift of the reference plane.

$$S_{11}' = e^{-2j\theta} S_{11}$$

This means that the phase of S_{11} is shifted by twice the electrical length of the shift in terminal plane of the second cable because the wave travels twice over this length upon incidence and reflection.

Directional Couplers

1. Following is schematic for measurement setup of a Directional Coupler with two different boundary conditions at the Input port (P1).

   (a) Open

   (b) 50 Ω matched

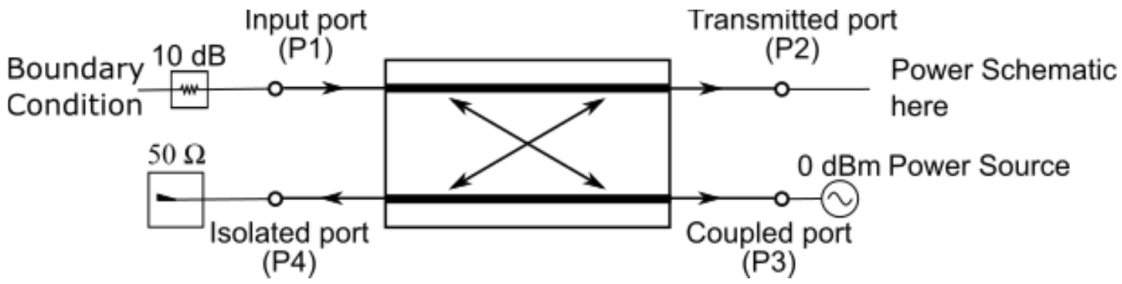


Figure 13: Schematic for Directional Coupler Measurement

Plot schematic of power at Port 2.

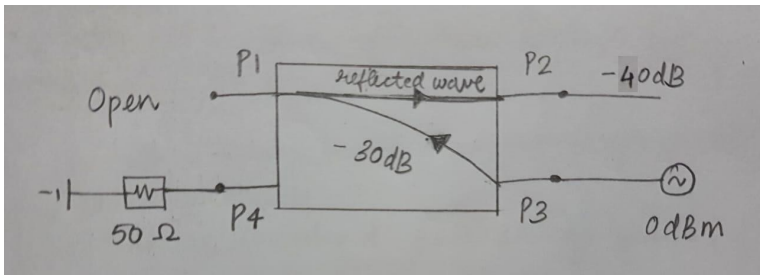


Figure 14: Power Schematic

(a) Open: Most of the power injected at P3 goes to P1 but since the P1 has an open boundary condition, the signal reflects in phase from P1 with an attenuation of 10 db and the attenuated signal goes to P2.

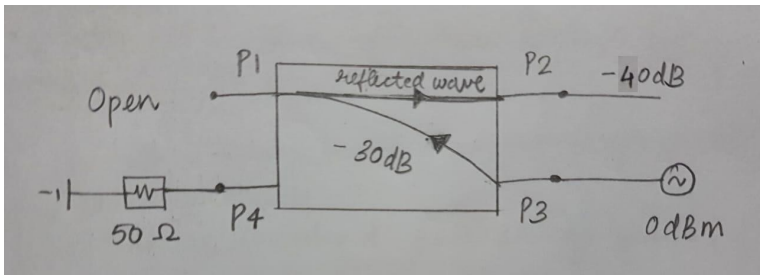


Figure 15: Power Schematic

(b) 50 Ω matched: This scheme is used for the measurement of the Directivity of the Coupler (S_{21} at the VNA) Most of the power goes to P1 and gets dissipated in the matched resistor. Ideally, there is no power received at P2 but practically about 0.01 % of power reaches P2.

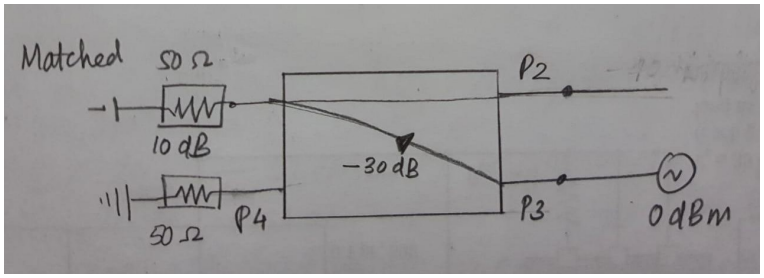


Figure 16: Power Schematic

2. In the following measurmeent setup for the directional coupler, the 50 termination of the Isolated port has been removed to keep the boundary condition as OPEN.

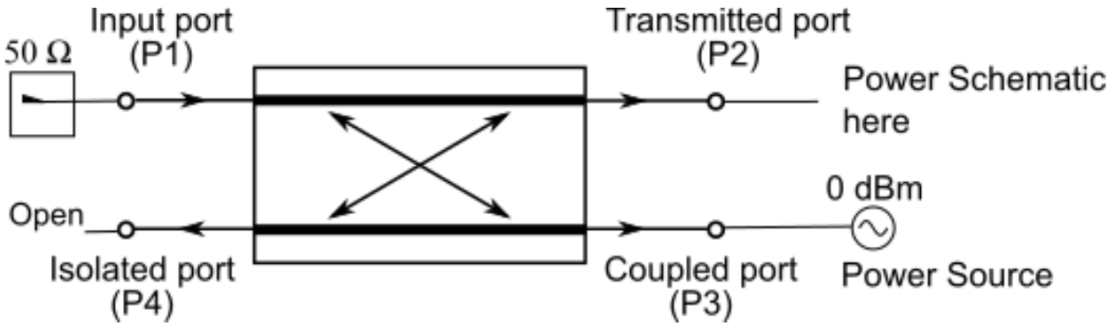


Figure 17: Schematic for Directional Coupler measurement with 50 matching removed

(a) Draw schematic of signal flow

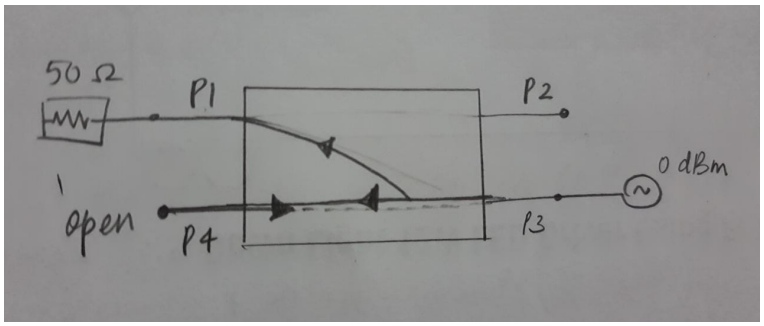


Figure 18: Power Schematic

(b) Plot the S parameters for the given measurement setup and explain the observations.

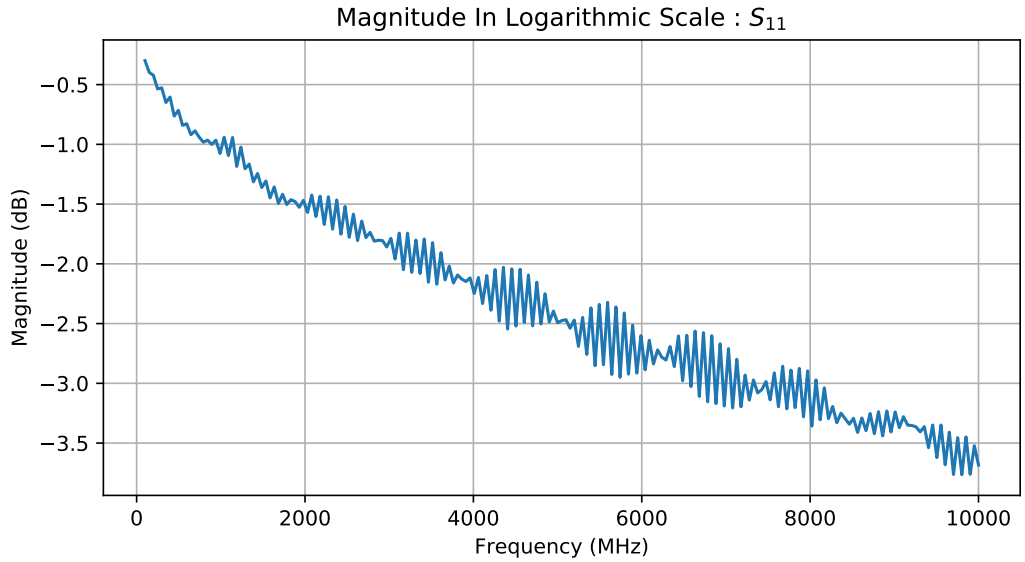


Figure 19: Configuration 1: Magnitude Plot of S_{11}

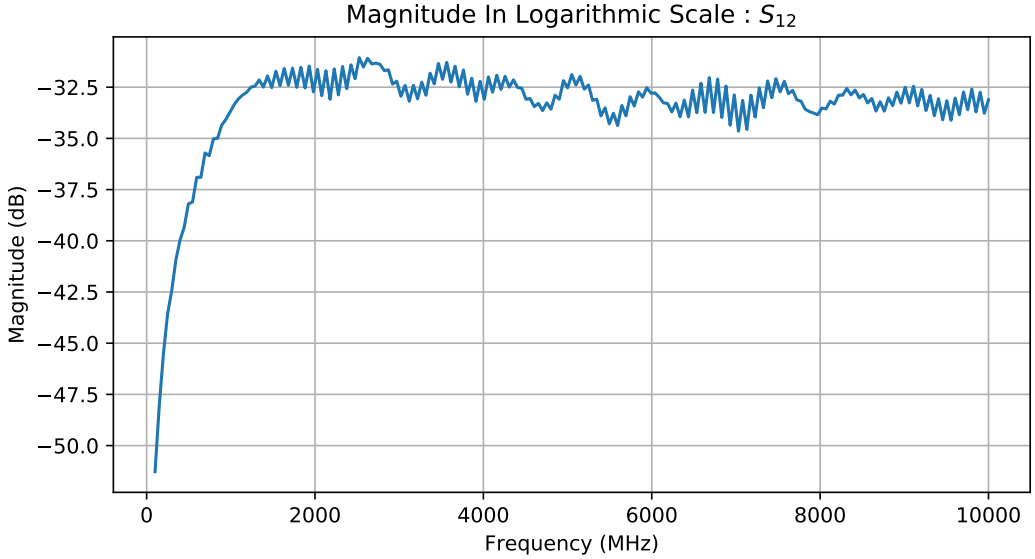


Figure 20: Configuration 1: Magnitude Plot of S_{12}

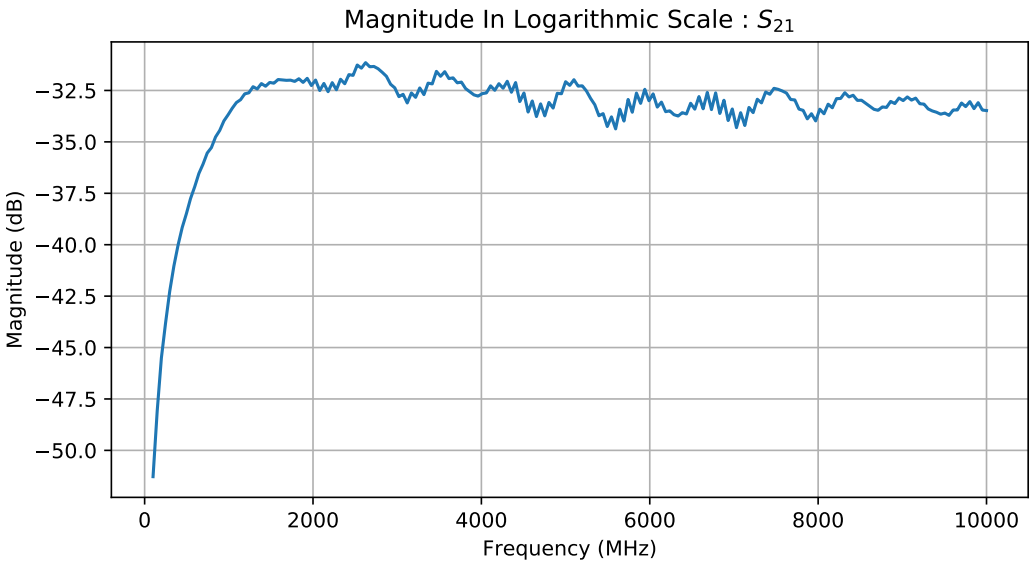


Figure 21: Configuration 1: Magnitude Plot of S_{21}

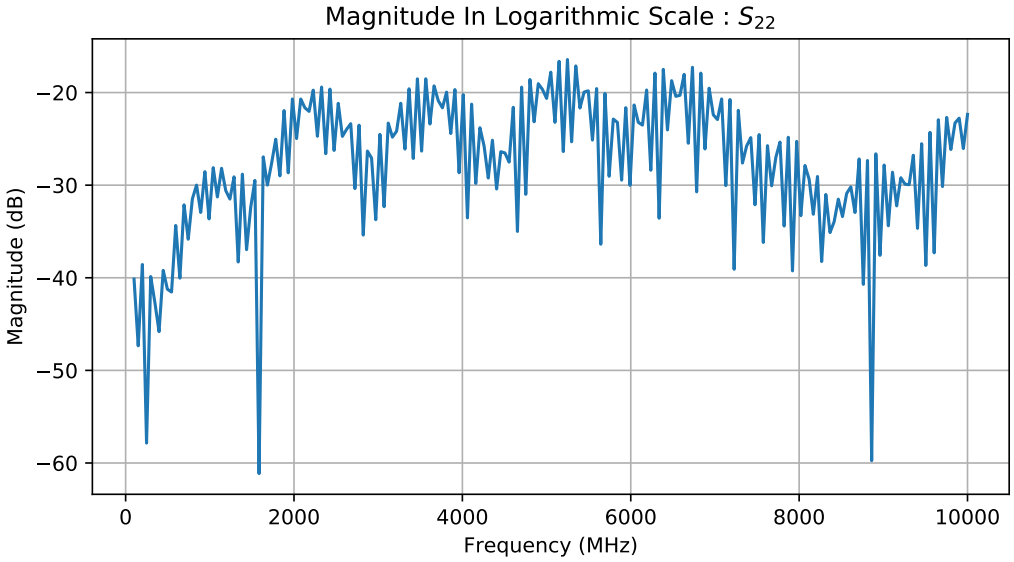


Figure 22: Configuration 1: Magnitude Plot of S_{22}

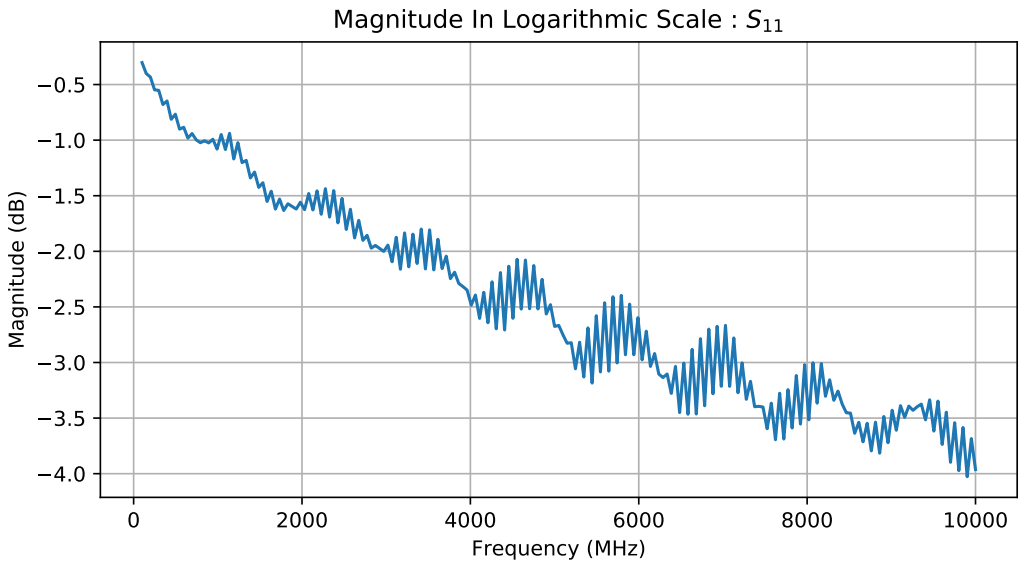


Figure 23: Configuration 2: Magnitude Plot of S_{11}

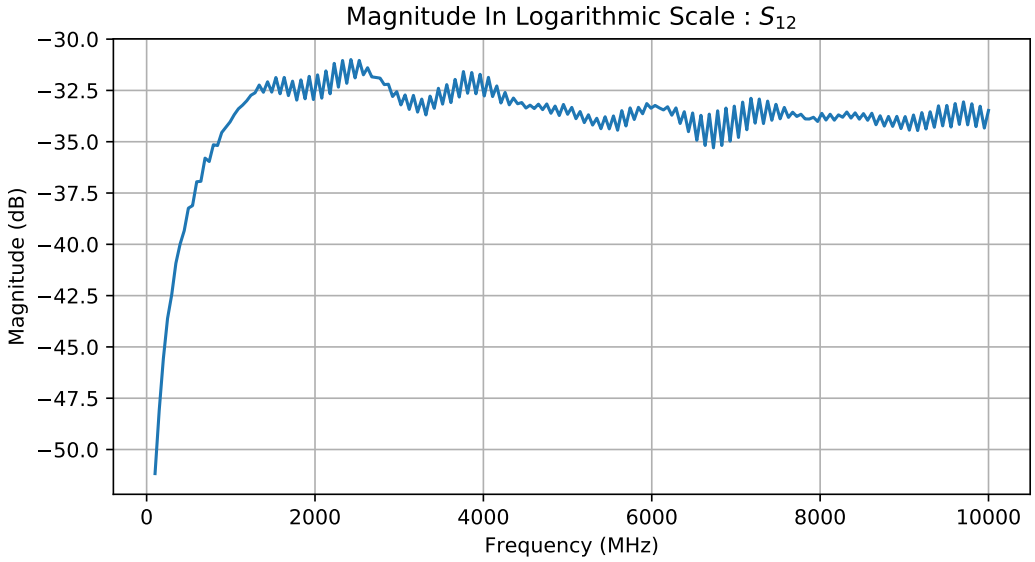


Figure 24: Configuration 2: Magnitude Plot of S_{12}

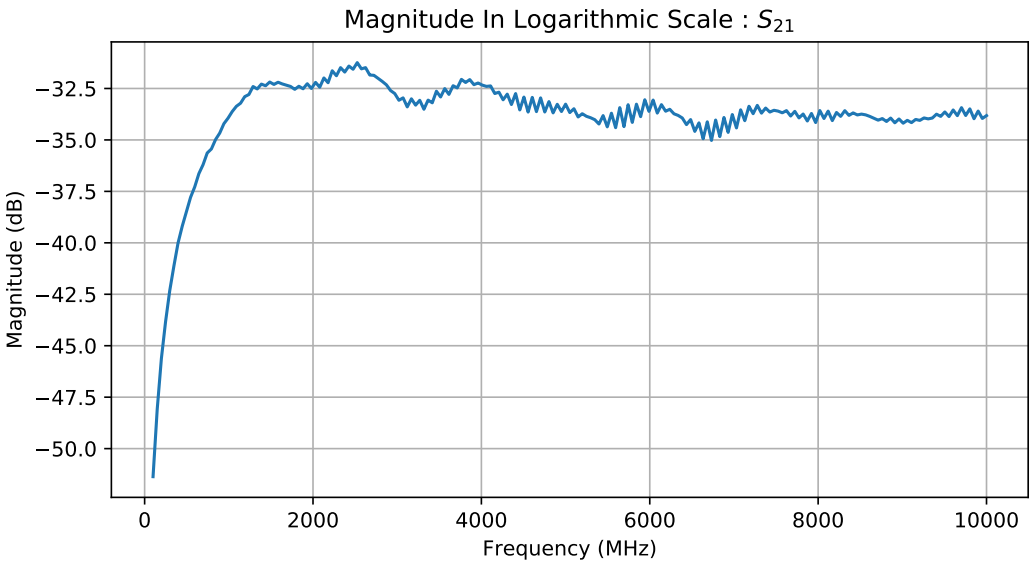


Figure 25: Configuration 2: Magnitude Plot of S_{21}

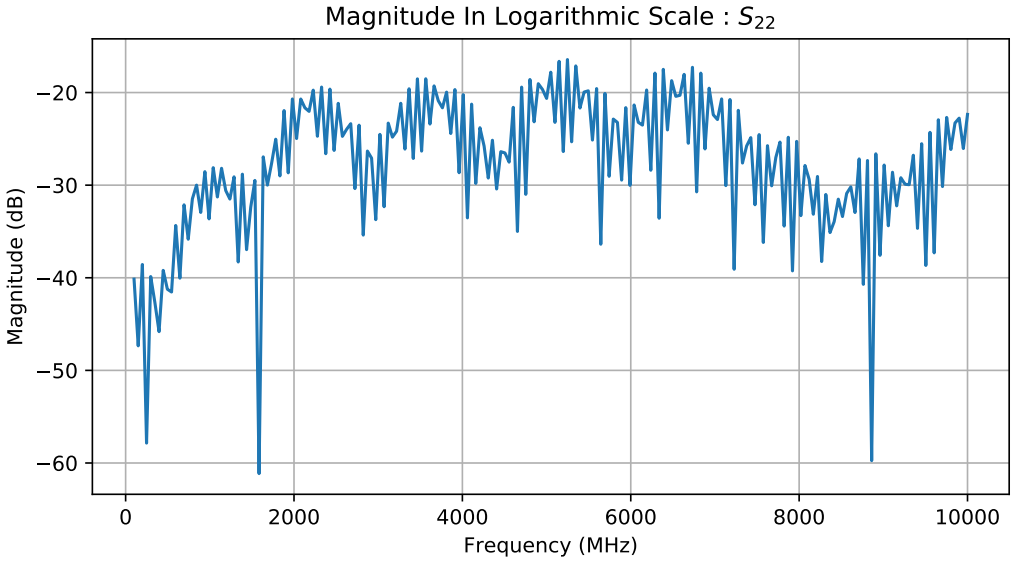


Figure 26: Configuration 2: Magnitude Plot of S_{22}

Observations

For both the configurations, the plots appear to be the same, so the directional coupler can be seen to be reversible in its usage. The reverse configuration gives the same results as the original configuration. Most of the power is reflected back to P3 and S_{11} is in the range of 0- -3 dB. Some of the power goes to P1 and that it can be seen that about -30 dB of power is obtained at P2, which is very little of the power injected.

References

[1] A. M. Nicolson, G. F. Ross, Measurement of the Intrinsic Properties of Materials by Time-Domain Techniques, IEEE Transactions on Instrumentation and Measurement, Vol. IM-19, No. 4, November 1970

[2] Adriano Luiz de Paula, Mirabel Cerqueira Rezende, Joaquim José Barroso, Experimental measurements and numerical simulation of permittivity and permeability of Teflon in X band, J. Aerosp.Technol. Manag., São José dos Campos, Vol.3, No.1, pp. 59-64, Jan. - Apr., 2011.

[3] David M. Pozar, Microwave Engineering, Wiley, November 2011.

Appendix: Codes

Vector Network Analyzer

Question 1

pages 4

ielecozar

```
plt.figure(1)
plt.plot(freq,phase_s11)
plt.title(' Phase Plot for $S_{11}$')
plt.ylabel('Phase in degrees ($^{\circ}$)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.show()
print(phase_s11)
```

Question 2

```
import matplotlib.pyplot as plt
import math
from math import *
import cmath
import numpy as np
import pandas as pd
from pandas import *
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('Data.csv', comment = '#', sep = ';')
frequency = data['freq[Hz]'].to_numpy()
#Frequency x axis
freq = frequency/1000000
#S11
real_s11 = data['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11 = data['im:Trc1_S11'].to_numpy(dtype=complex)
s11 = real_s11 + imag_s11*1j
mag_s11 = 20*np.log10(np.absolute(s11))
ph_s11 = np.angle(s11, deg=1)
#S21
real_s21 = data['re:Trc3_S21'].to_numpy(dtype=complex)
imag_s21 = data['im:Trc3_S21'].to_numpy(dtype=complex)
s21 = real_s21 + imag_s21*1j
mag_s21 = 20*np.log10(np.absolute(s21))
ph_s21 = np.angle(s21, deg=1)
#The parameter K
K = (1-(s21**2 - s11**2))/(2*s11)

Gamma = K + np.sqrt(K**2-1)
for i in range(len(Gamma)):
    if (np.absolute(Gamma[i])>1):
        Gamma[i]=K[i]-np.sqrt(K[i]**2-1)

T = ((s11+s21)-Gamma)/(1 -(s11+s21)*Gamma)
x = ((1+Gamma)/(1-Gamma))**2
w = 2*math.pi*frequency
c = 2.998*(10**8)
d = ((10.67-7.92)/2)*10**(-3)
d = 0.00275/2
y = -((c/(w*d)*np.log(1/T)))**2
Er = (y/x)**0.5
dielectric_constant = np.real(Er)
```

```
#Plotting the data

plt.figure(1)
plt.plot(freq,dielectric_constant,linestyle='-')
plt.title('Dielectric Constant')
plt.ylabel('Dielectric Constant, $Re(\epsilon_r)$')
plt.xlabel('Frequency (MHz)')
plt.grid(True,which='both')

plt.figure(2)
plt.plot(freq,np.absolute(Gamma))
plt.title('Reflection Coefficient versus Frequency plot')
plt.ylabel('Magnitude of Reflection Coefficient, |\Gamma|')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.show()
```

Question 3

```
import matplotlib.pyplot as plt
import math
from math import *
import cmath
import numpy as np
import pandas as pd
from pandas import *
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('Data.csv', comment = '#', sep = ';')
frequency = data['freq[Hz]'].to_numpy()
#Frequency x axis
freq = frequency/1000000
#S11
real_s11 = data['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11 = data['im:Trc1_S11'].to_numpy(dtype=complex)
s11 = real_s11 + imag_s11*1j
mags11 = np.absolute(s11)
dbs11 = 20*np.log10(mags11)
#S21
real_s21 = data['re:Trc3_S21'].to_numpy(dtype=complex)
imag_s21 = data['im:Trc3_S21'].to_numpy(dtype=complex)
s21 = real_s21 + imag_s21*1j
mags21 = np.absolute(s21)
dbs21 = 20*np.log10(mags21)
#S12
real_s12 = data['re:Trc2_S12'].to_numpy(dtype=complex)
imag_s12 = data['im:Trc2_S12'].to_numpy(dtype=complex)
s12 = real_s12 + imag_s12*1j
mags12 = np.absolute(s12)
dbs12 = 20*np.log10(mags12)
#S22
real_s22 = data['re:Trc4_S22'].to_numpy(dtype=complex)
imag_s22 = data['im:Trc4_S22'].to_numpy(dtype=complex)
s22 = real_s22 + imag_s22*1j
mags22 = np.absolute(s22)
dbs22 = 20*np.log10(mags22)
#Plotting the data
plt.figure(1)
plt.plot(freq, dbs11,'b')
plt.title('Magnitude In Logarithmic Scale : $S_{11}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
```

```
plt.grid(True)

plt.plot(freq, dbs12,'b')
plt.title('Magnitude In Logarithmic Scale : $S_{12}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.plot(freq, dbs21,'b')
plt.title('Magnitude In Logarithmic Scale : $S_{21}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.plot(freq, dbs22,'b')
plt.title('Magnitude In Logarithmic Scale : $S_{22}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.show()
```

Question 4

```
import matplotlib.pyplot as plt
import math
from math import *
import cmath
import numpy as np
import pandas as pd
from pandas import *
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('Data.csv', comment = '#', sep = ';')
frequency = data['freq[Hz]'].to_numpy()
#Frequency x axis
freq = frequency/1000000
#S11
real_s11 = data['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11 = data['im:Trc1_S11'].to_numpy(dtype=complex)
s11 = real_s11 + imag_s11*1j
mag_s11 = 20*np.log10(np.absolute(s11))
ph_s11 = np.angle(s11, deg=1)
#S21
real_s21 = data['re:Trc3_S21'].to_numpy(dtype=complex)
imag_s21 = data['im:Trc3_S21'].to_numpy(dtype=complex)
s21 = real_s21 + imag_s21*1j
mag_s21 = 20*np.log10(np.absolute(s21))
ph_s21 = np.angle(s21, deg=1)
#The parameter K
K = (1-(s21**2 - s11**2))/(2*s11)

Gamma = K + np.sqrt(K**2-1)
for i in range(len(Gamma)):
    if (np.absolute(Gamma[i])>1):
        Gamma[i]=K[i]-np.sqrt(K[i]**2-1)

T = ((s11+s21)-Gamma)/(1 -(s11+s21)*Gamma)
x = ((1+Gamma)/(1-Gamma))**2
w = 2*math.pi*frequency
c = 2.998*(10**8)
d = ((10.67-7.92)/2)*10**(-3)
d = 0.00275/2
y = -((c/(w*d)*np.log(1/T)))**2
Er = (y/x)**0.5
dielectric_constant = np.real(Er)
Er_real = np.real(Er)
```

```
Er_imag = np.imag(Er)
loss_tan = np.tan(Er_real/Er_imag)
#Plotting the data
plt.figure(1)
plt.plot(freq,np.absolute(T))
plt.title('Transmission Coefficient versus Frequency plot')
plt.ylabel('Magnitude of Transmission Coefficient, |T|')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.figure(2)
plt.plot(freq,Er_imag)
plt.title('Loss tangent versus Frequency plot')
plt.ylabel('Loss Tangent')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.show()
```

Question 5

```
import matplotlib.pyplot as plt
import cmath
import numpy as np
import pandas as pd
from pandas import *
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data_2ft = pd.read_csv('Data_2ft.csv', comment = '#', sep = ';')
frequency = data_2ft['freq[Hz]'].to_numpy()
freq = frequency/1000000
real_s11_2ft = data_2ft['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11_2ft = data_2ft['im:Trc1_S11'].to_numpy(dtype=complex)
s11_2ft = real_s11_2ft + imag_s11_2ft*1j
phase_s11_2ft = np.angle(s11_2ft, deg = 1)

data_12ft = pd.read_csv('Data_12ft.csv', comment = '#', sep = ';')
real_s11_12ft = data_12ft['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11_12ft = data_12ft['im:Trc1_S11'].to_numpy(dtype=complex)
s11_12ft = real_s11_12ft + imag_s11_12ft*1j
phase_s11_12ft = np.angle(s11_12ft, deg = 1)

#Plotting the data
plt.figure(1)
plt.plot(freq,phase_s11_2ft)
plt.title(' Phase Plot for $S_{11}$: 2 ft Cable')
plt.ylabel('Phase in degrees ($^{\circ}$)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.figure(2)
plt.plot(freq,phase_s11_12ft)
plt.title(' Phase Plot for $S_{11}$: 12 ft Cable')
plt.ylabel('Phase in degrees ($^{\circ}$)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

plt.show()
```

Directional COupler

Question 2: Configuration 1

```
import matplotlib.pyplot as plt
import math
from math import *
import cmath
import numpy as np
import pandas as pd
from pandas import *
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('DC1.csv', comment = '#', sep = ';')
frequency = data['freq[Hz]'].to_numpy()
#Frequency x axis
freq = frequency/1000000
#S11
real_s11 = data['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11 = data['im:Trc1_S11'].to_numpy(dtype=complex)
s11 = real_s11 + imag_s11*1j
mags11 = np.absolute(s11)
dbs11 = 20*np.log10(mags11)
phases11 = np.angle(s11)
#S21
real_s21 = data['re:Trc3_S21'].to_numpy(dtype=complex)
imag_s21 = data['im:Trc3_S21'].to_numpy(dtype=complex)
s21 = real_s21 + imag_s21*1j
mags21 = np.absolute(s21)
dbs21 = 20*np.log10(mags21)
phases21 = np.angle(s21)
#S12
real_s12 = data['re:Trc2_S12'].to_numpy(dtype=complex)
imag_s12 = data['im:Trc2_S12'].to_numpy(dtype=complex)
s12 = real_s12 + imag_s12*1j
mags12 = np.absolute(s12)
dbs12 = 20*np.log10(mags12)
phases12 = np.angle(s12)
#S22
real_s22 = data['re:Trc4_S22'].to_numpy(dtype=complex)
imag_s22 = data['im:Trc4_S22'].to_numpy(dtype=complex)
s22 = real_s22 + imag_s22*1j
mags22 = np.absolute(s22)
dbs22 = 20*np.log10(mags22)
phases22 = np.angle(s22)
```

```
#Plotting the data
plt.figure(1)
plt.plot(freq, dbs11)
plt.title('Magnitude In Logarithmic Scale : $S_{11}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.figure(2)
plt.plot(freq, dbs12)
plt.title('Magnitude In Logarithmic Scale : $S_{12}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.figure(3)
plt.plot(freq, dbs21)
plt.title('Magnitude In Logarithmic Scale : $S_{21}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.figure(4)
plt.plot(freq, dbs22)
plt.title('Magnitude In Logarithmic Scale : $S_{22}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.show()
```

Question 2: Configuration 2

```
import matplotlib.pyplot as plt
import math
from math import *
import cmath
import numpy as np
import pandas as pd
from pandas import *
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('DC2.csv', comment = '#', sep = ';')
frequency = data['freq[Hz]'].to_numpy()
#Frequency x axis
freq = frequency/1000000
#S11
```

```
real_s11 = data['re:Trc1_S11'].to_numpy(dtype=complex)
imag_s11 = data['im:Trc1_S11'].to_numpy(dtype=complex)
s11 = real_s11 + imag_s11*1j
mags11 = np.absolute(s11)
dbs11 = 20*np.log10(mags11)
phases11 = np.angle(s11)
#S21
real_s21 = data['re:Trc3_S21'].to_numpy(dtype=complex)
imag_s21 = data['im:Trc3_S21'].to_numpy(dtype=complex)
s21 = real_s21 + imag_s21*1j
mags21 = np.absolute(s21)
dbs21 = 20*np.log10(mags21)
phases21 = np.angle(s21)
#S12
real_s12 = data['re:Trc2_S12'].to_numpy(dtype=complex)
imag_s12 = data['im:Trc2_S12'].to_numpy(dtype=complex)
s12 = real_s12 + imag_s12*1j
mags12 = np.absolute(s12)
dbs12 = 20*np.log10(mags12)
phases12 = np.angle(s12)
#S22
real_s22 = data['re:Trc4_S22'].to_numpy(dtype=complex)
imag_s22 = data['im:Trc4_S22'].to_numpy(dtype=complex)
s22 = real_s22 + imag_s22*1j
mags22 = np.absolute(s22)
dbs22 = 20*np.log10(mags22)
phases22 = np.angle(s22)

#Plotting the data
plt.figure(1)
plt.plot(freq, dbs11)
plt.title('Magnitude In Logarithmic Scale : $S_{11}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.figure(2)
plt.plot(freq, dbs12)
plt.title('Magnitude In Logarithmic Scale : $S_{12}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.figure(3)
plt.plot(freq, dbs21)
plt.title('Magnitude In Logarithmic Scale : $S_{21}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
```

```
plt.grid(True)
plt.figure(4)
plt.plot(freq, dbs22)
plt.title('Magnitude In Logarithmic Scale : $S_{22}$')
plt.ylabel('Magnitude (dB)')
plt.xlabel('Frequency (MHz)')
plt.grid(True)
plt.show()
```

INDIAN INSTITUTE OF SCIENCE

IISC QUANTUM TECHNOLOGY INITIATIVE

QT211 - BASIC QUANTUM TECHNOLOGY LABORATORY

Assignment 3

Chaitali Shah
(SR No: 01-02-04-10-51-21-1-19786)

Date: December 12, 2021

Aim of the Experiment

1. Resonators
    (a) To Fit the S parameters
    (b) To calculate the quality factors and the resonance frequency for each resonator
    (c) To report the χ^2 of the fit.
2. Qiskit
    (a) To create Bell states and visualise them
    (b) To show Gate Identities
3. Rabi Oscillations
    (a) Solve the Rabi Problem for σ_y
    (b) Find the Transition frequency f_{12} of the transmon
        i. Using Spectroscopy (Sweeping Frequency)
        ii. Using Rabi Oscillations (Sweeping Amplitude)

Resonators

1. Fit S parameters to find resonant frequency, quality factor and χ^2 of the fit.

Resonator 1

The magnitude of S_{21} is plotted below. Curve-fitting is implemented using the Lorentzian function and the fitted plot is also shown.

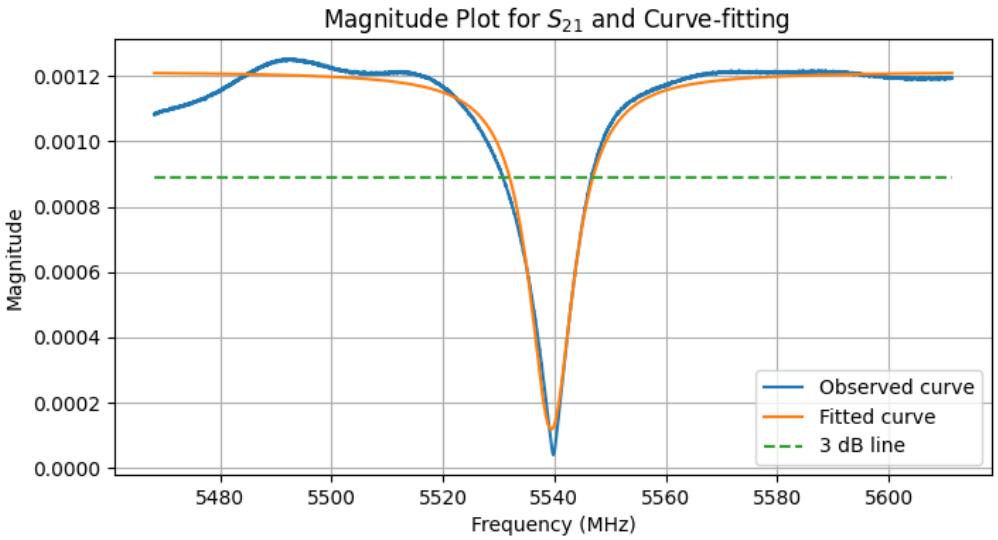


Figure 1: The Magnitude plot and fitting for Resonator 1

The resonant frequency is observed to be $f_0 = 5539.44$ Mhz ≈ 5540 MHz.
The bandwidth is found by observing the frequencies at which the magnitude plot intersects the 3dB line (magnitude becomes $\frac{S_{21,f_0}}{\sqrt{2}}$).
$f_1 = 5546.93$ MHz, $f_2 = 4431.93$ MHz

$$BW = \Delta f = f_1 - f_2 = 5546.93 - 4431.93 \text{ MHz}$$
$$BW = \Delta f = 15 \text{ MHz}$$

The Quality factor is computed to be,

$$Q = \frac{f_0}{\Delta f} = \frac{5540}{15} = \frac{1108}{3}$$
$$Q = 369.33$$

The χ^2 value is found using the formula:

$$\chi^2 = \sum_{n=1}^{N} \frac{(Observed - Expected)^2}{Expected}$$

The χ^2 value is calculated to be 0.00875.

Resonator 2

The magnitude of S_{21} is plotted for different power settings.

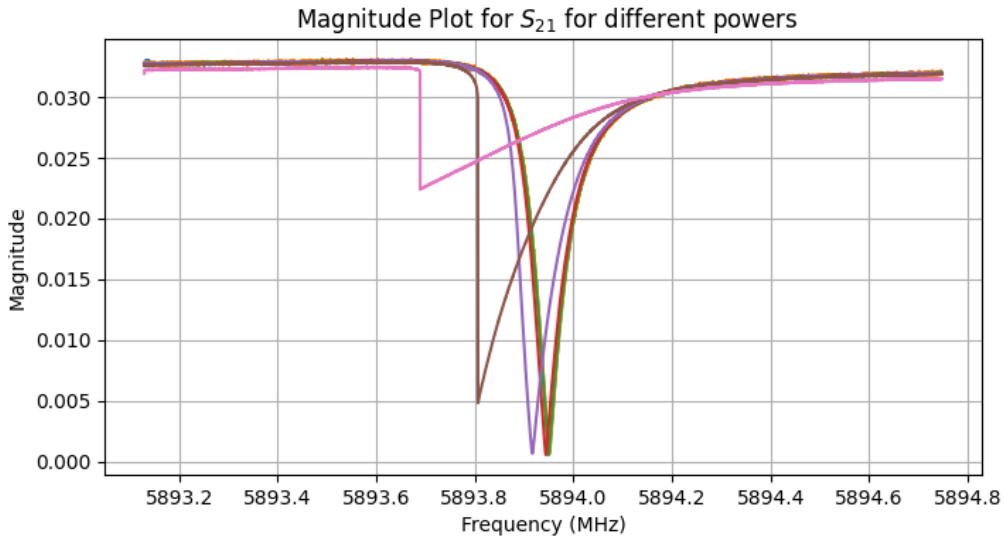


Figure 2: The Magnitude plots for Resonator 2

Curve-fitting is implemented using the Lorentzian function and the fitted plot is also shown.

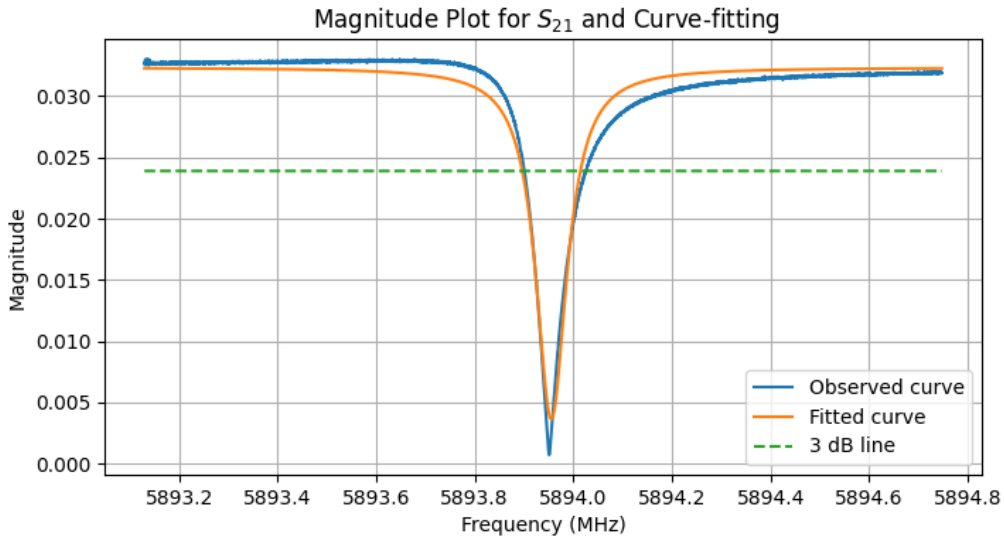


Figure 3: The Magnitude plot and curve-fitting for Resonator 2

The resonant frequency is observed to be $f_0 = 5893.95$ Mhz.
The bandwidth is found by observing the frequencies at which the magnitude plot intersects the 3dB line (magnitude becomes $\frac{S_{21,f_0}}{\sqrt{2}}$).
$f_1 = 5894.01$ MHz, $f_2 = 5893.89$ MHz

$$BW = \Delta f = f_1 - f_2 = 5893.89 - 5894.01 \text{ MHz}$$
$$BW = \Delta f = 0.12 \text{ MHz}$$

The Quality factor is computed to be,

$$Q = \frac{f_0}{\Delta f} = \frac{5893.95}{0.12} = \frac{196465}{4}$$
$$Q = 49116.25$$

The χ^2 value is found using the formula:

$$\chi^2 = \sum_{n=1}^{N} \frac{(Observed - Expected)^2}{Expected}$$

The χ^2 value is calculated to be 0.25809.

Resonator 3

The magnitude of S_{21} is plotted for different power settings.

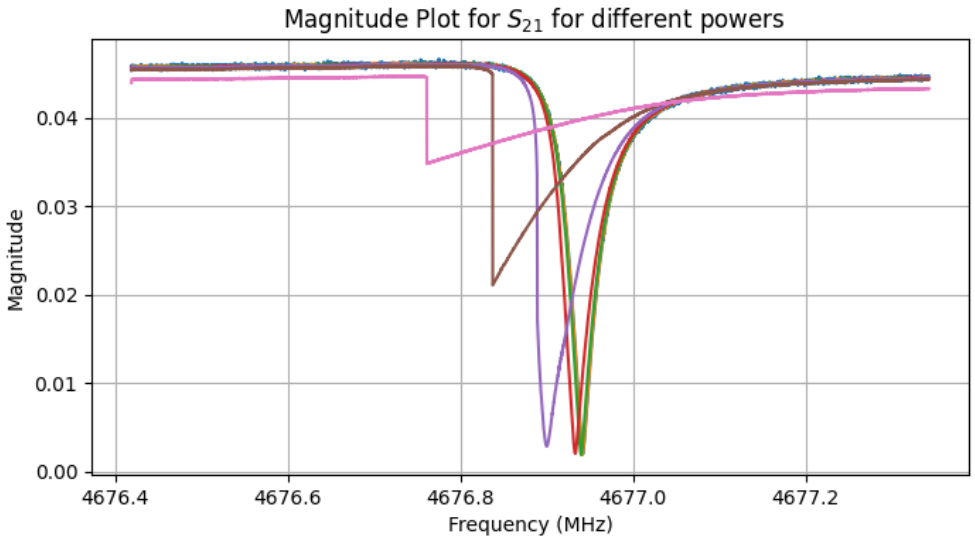


Figure 4: The Magnitude plots for Resonator 3

Curve-fitting is implemented using the Lorentzian function and the fitted plot is also shown.

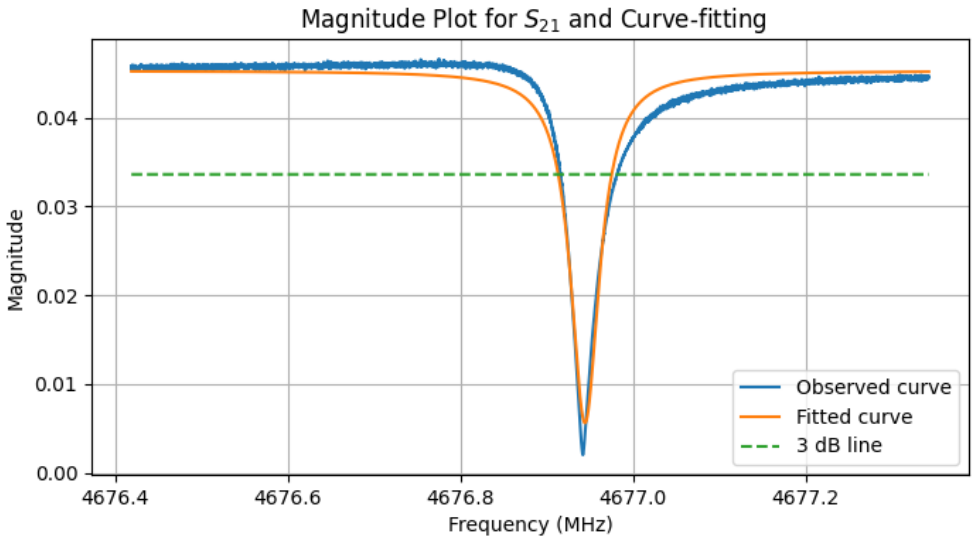


Figure 5: The Magnitude plot and curve-fitting for Resonator 3

The resonant frequency is observed to be $f_0 = 4676.944$ Mhz.
The bandwidth is found by observing the frequencies at which the magnitude plot intersects the 3dB line (magnitude becomes $\frac{S_{21,f_0}}{\sqrt{2}}$).
$f_1 = 4676.97$ MHz, $f_2 = 4676.91$ MHz

$$BW = \Delta f = f_1 - f_2 = 4676.97 - 4676.91 \text{ MHz}$$
$$BW = \Delta f = 0.06 \text{ MHz}$$

The Quality factor is computed to be,

$$Q = \frac{f_0}{\Delta f} = \frac{4676.944}{0.06}$$
$$Q = 77949.0666$$

The χ^2 value is found using the formula:

$$\chi^2 = \sum_{n=1}^{N} \frac{(Observed - Expected)^2}{Expected}$$

The χ^2 value is calculated to be 0.35148.

Resonator 4

The magnitude of S_{21} is plotted for different power settings.

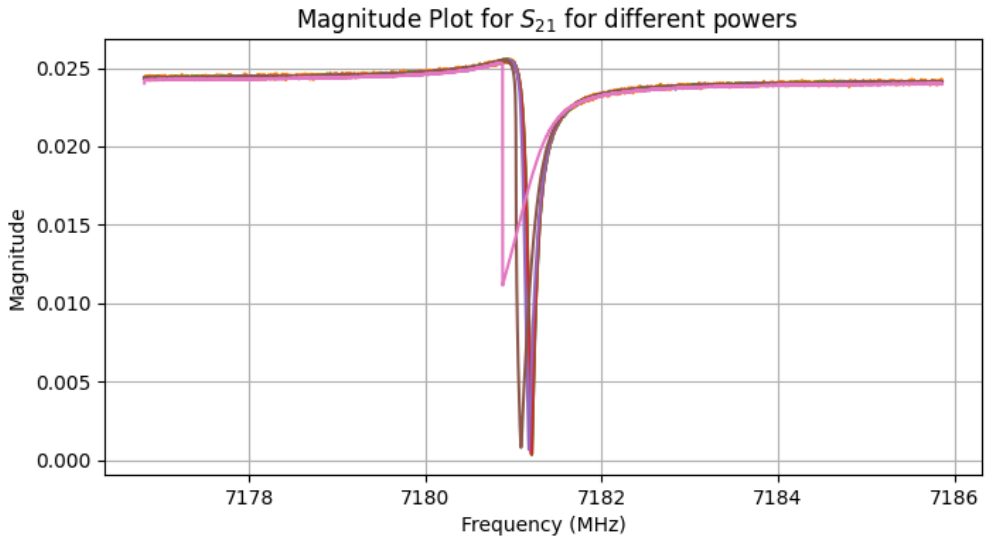


Figure 6: The Magnitude plots for Resonator 4

Curve-fitting is implemented using the Lorentzian function and the fitted plot is also shown.

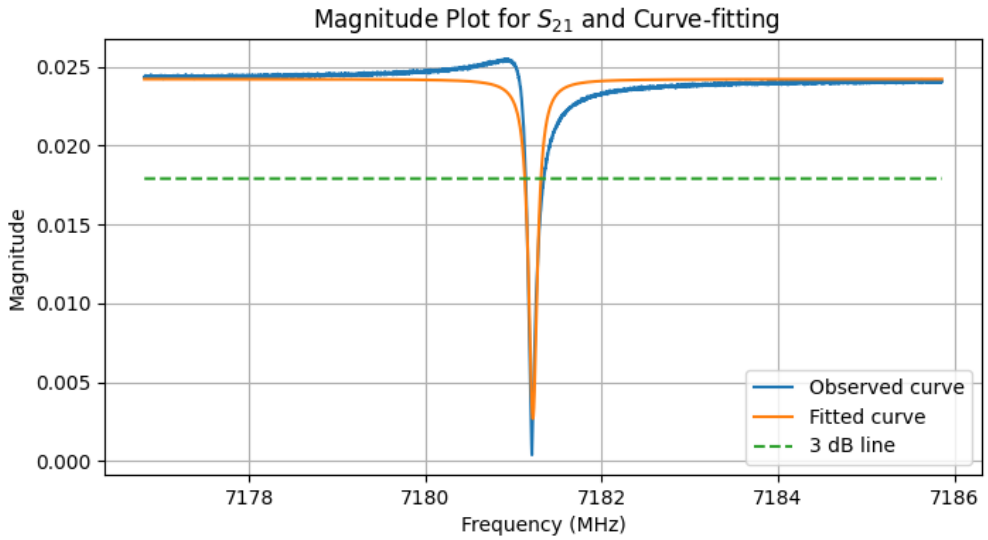


Figure 7: The Magnitude plot and curve-fitting for Resonator 4

The resonant frequency is observed to be $f_0 = 7181.2200$ Mhz.
The bandwidth is found by observing the frequencies at which the magnitude plot intersects the 3dB line (magnitude becomes $\frac{S_{21,f_0}}{\sqrt{2}}$).
$f_1 = 7181.3112$ MHz, $f_2 = 7181.1284$ MHz

$$BW = \Delta f = f_1 - f_2 = 7181.3112 - 7181.1284 \text{ MHz}$$
$$BW = \Delta f = 0.1828 \text{ MHz}$$

The Quality factor is computed to be,

$$Q = \frac{f_0}{\Delta f} = \frac{7181.22}{0.1828}$$
$$Q = 39284.5733$$

The χ^2 value is found using the formula:

$$\chi^2 = \sum_{n=1}^{N} \frac{(Observed - Expected)^2}{Expected}$$

The χ^2 value is calculated to be 0.1576.

Qiskit

1. Create the following Bell States and visualise them.

   (a) $|00\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

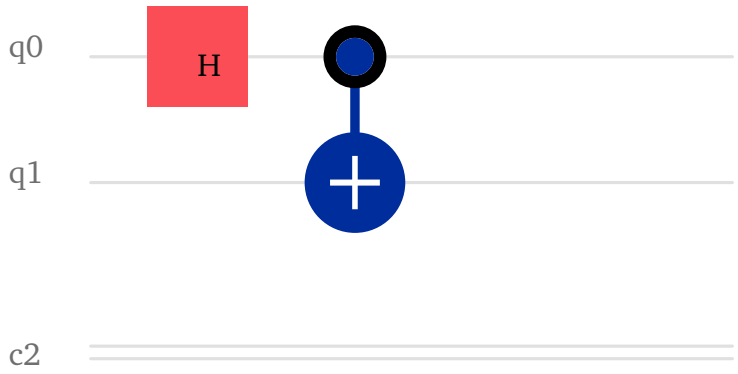


Figure 8: Quantum Circuit for Bell State $\left|\phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

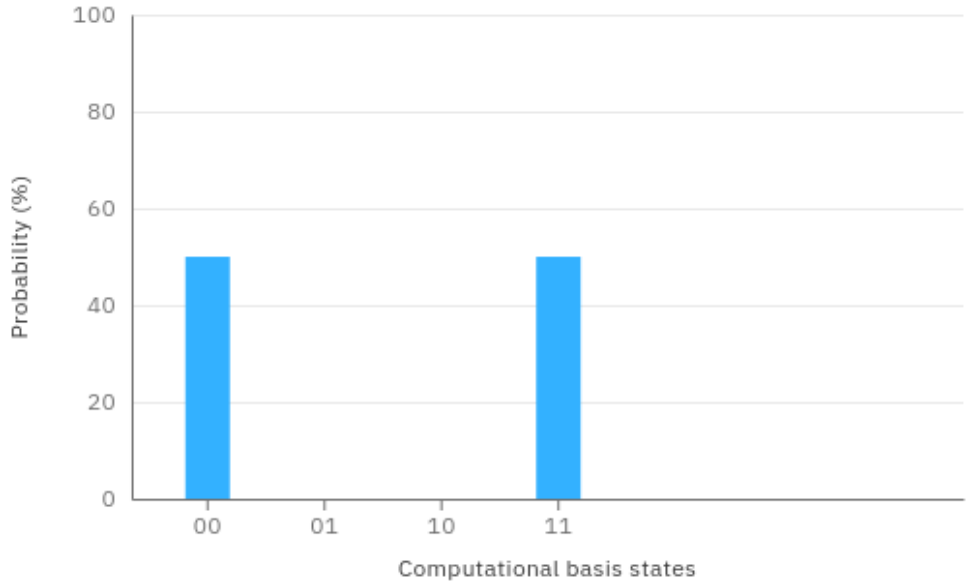


Figure 9: Probability distribution of Bell State $\left|\phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

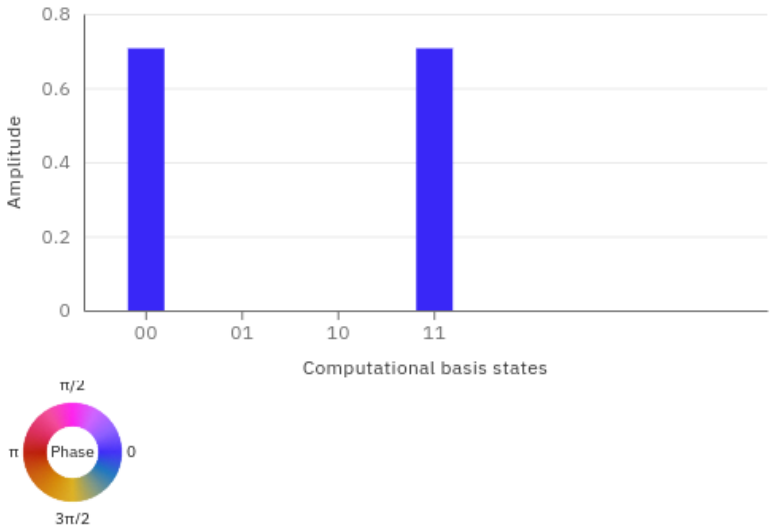


Figure 10: Amplitudes of states for Bell State $\left|\phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

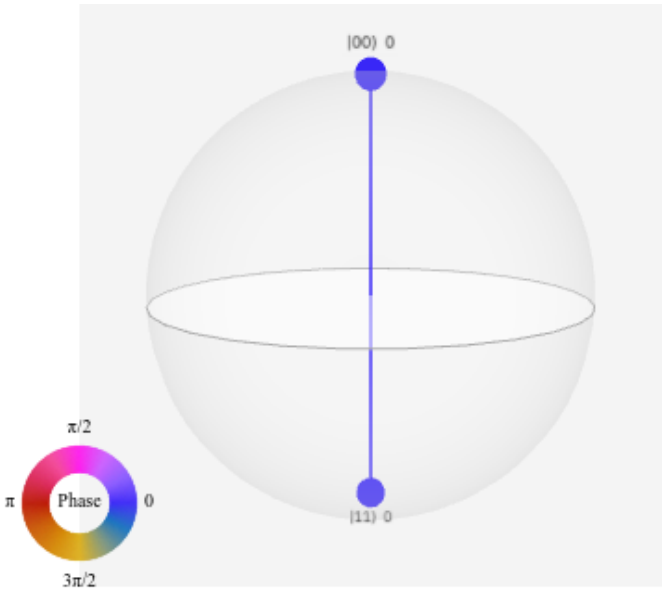


Figure 11: Visualisation of Bell State $\left|\phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ in Q-sphere

(b) $|01\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$

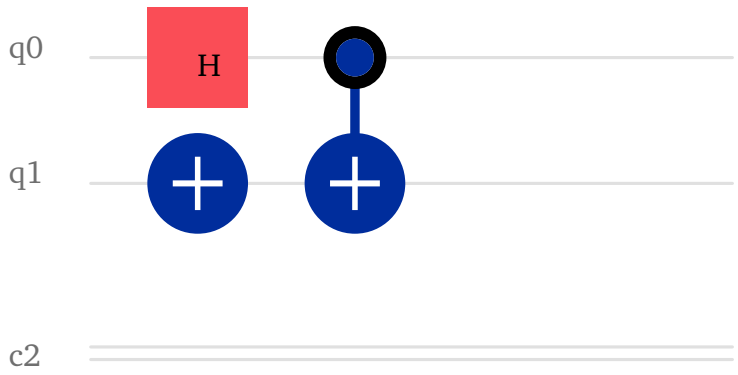


Figure 12: Quantum Circuit for Bell State $\left|\psi^+\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$

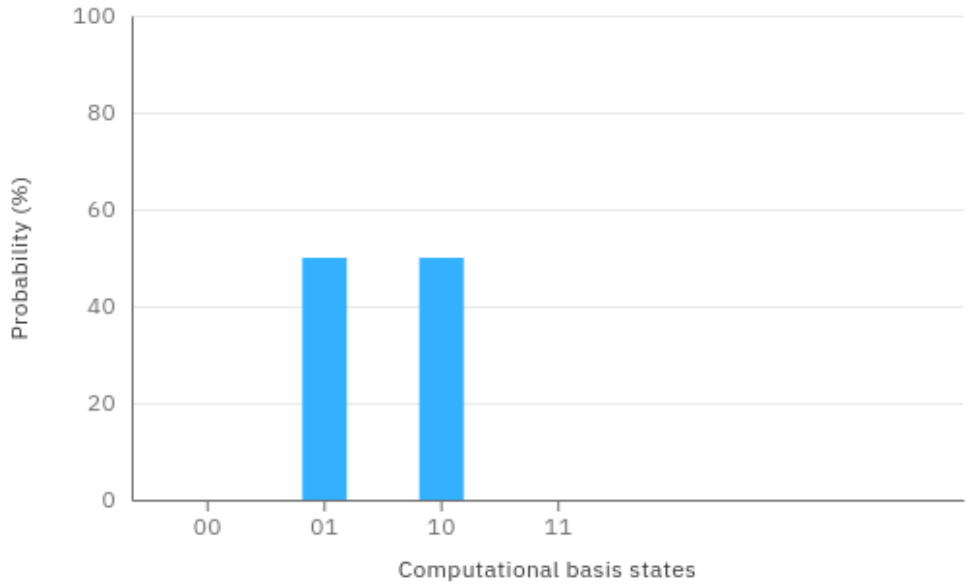


Figure 13: Probability distribution of Bell State $\left|\psi^+\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$

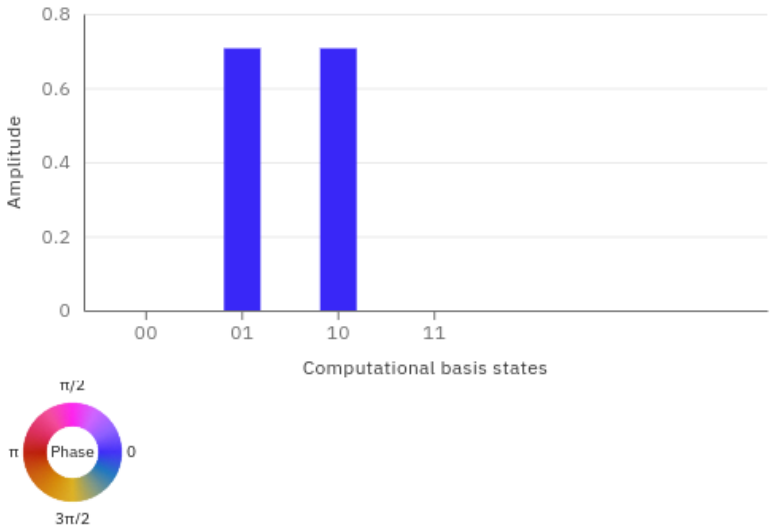


Figure 14: Amplitudes of states for Bell State $\left|\psi^{+}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$

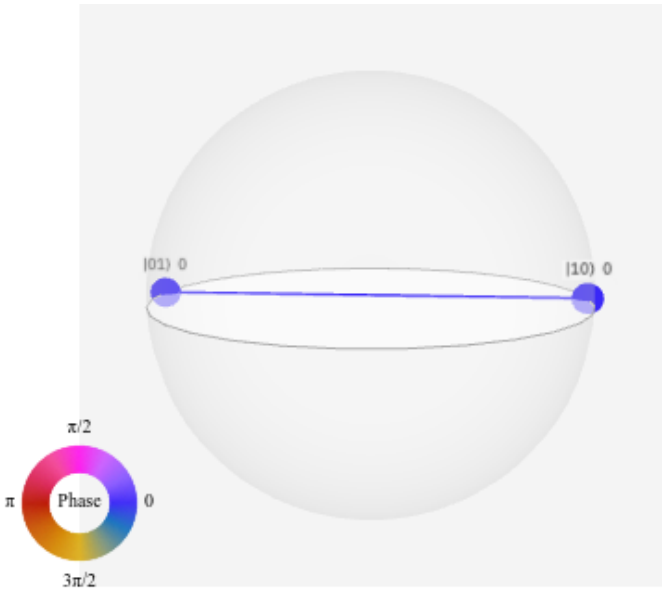


Figure 15: Visualisation of Bell State $\left|\psi^{+}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$ in Q-sphere

(c) $|10\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$

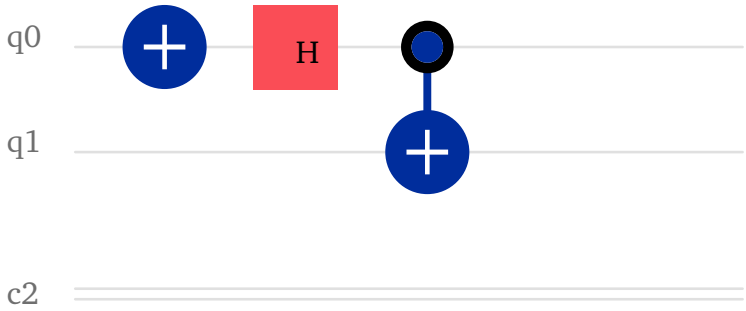


Figure 16: Quantum Circuit for Bell State $\left|\phi^-\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$

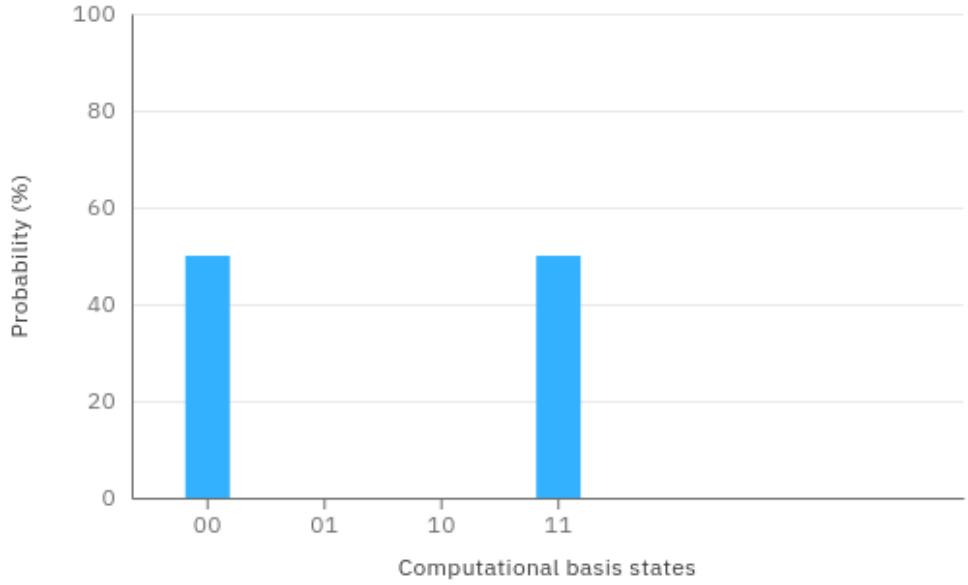


Figure 17: Probability distribution of Bell State $\left|\phi^-\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$

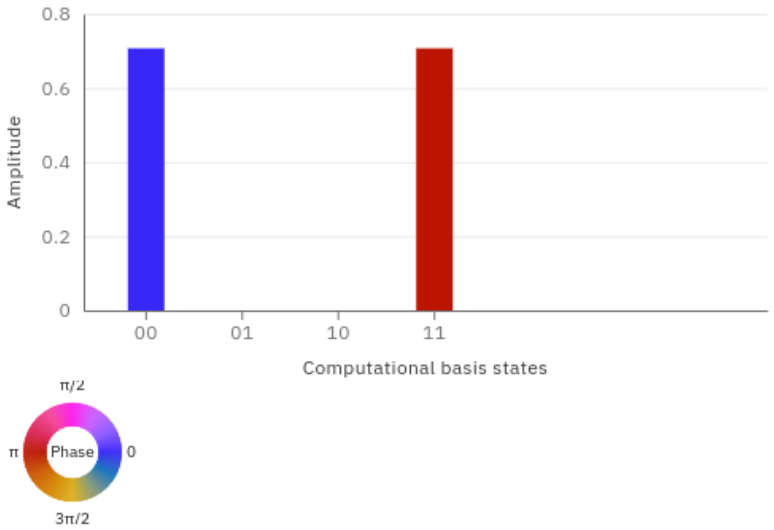


Figure 18: Amplitudes of states for Bell State $\left|\phi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$

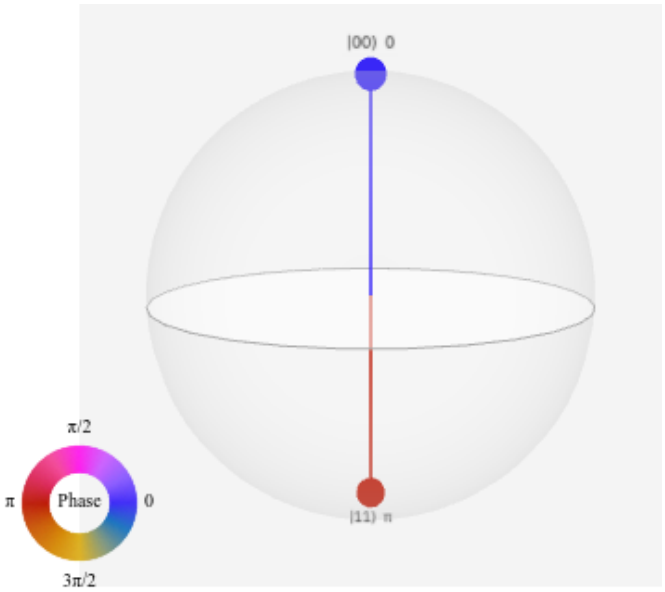


Figure 19: Visualisation of Bell State $\left|\phi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$ in Q-sphere

(d) $|11\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$

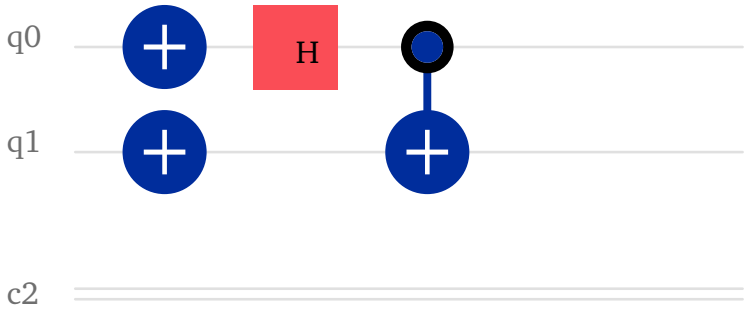


Figure 20: Quantum Circuit for Bell State $\left|\psi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$

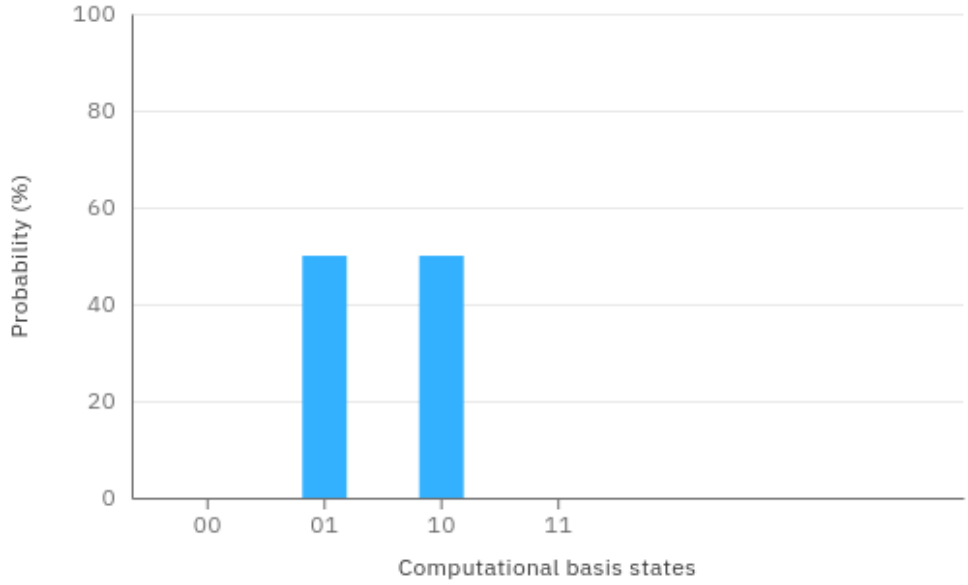


Figure 21: Probability distribution of Bell State $\left|\psi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$

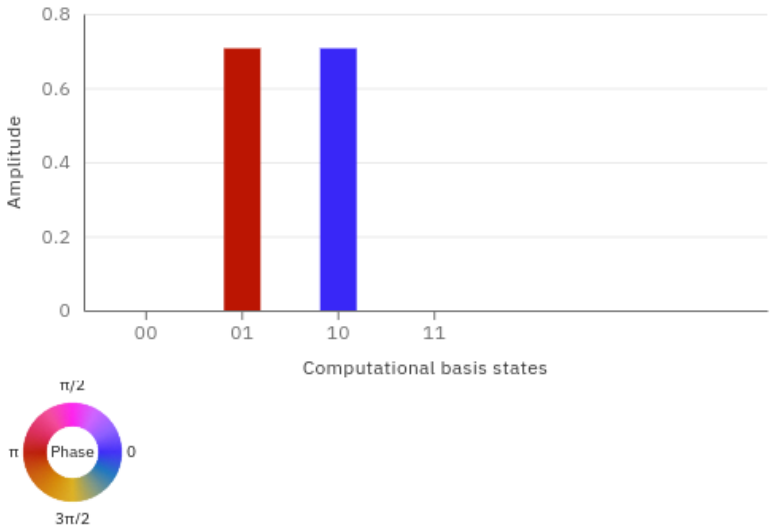


Figure 22: Amplitudes of states for Bell State $\left|\psi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$

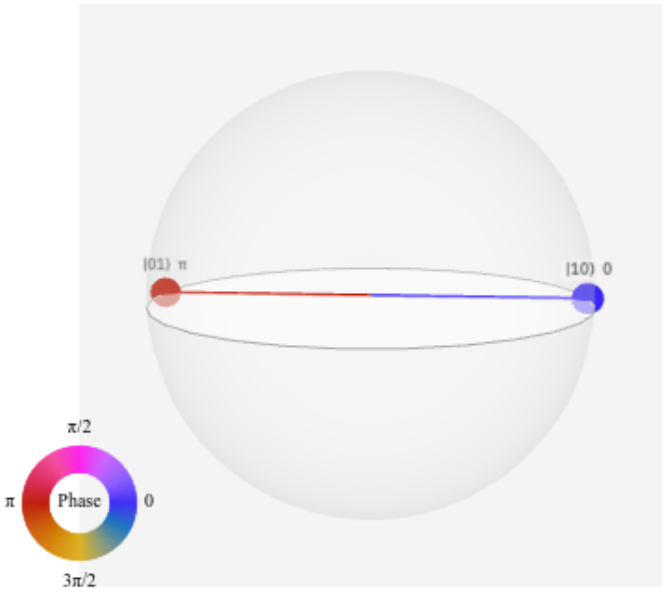


Figure 23: Visualisation of Bell State $\left|\psi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$ in Q-sphere

2. Show the following Gate identities
   (Note: All quantum registers are initialised to $|0\rangle$)

   (a) HXH = Z

Figure 24: LHS = HXH$|0\rangle$

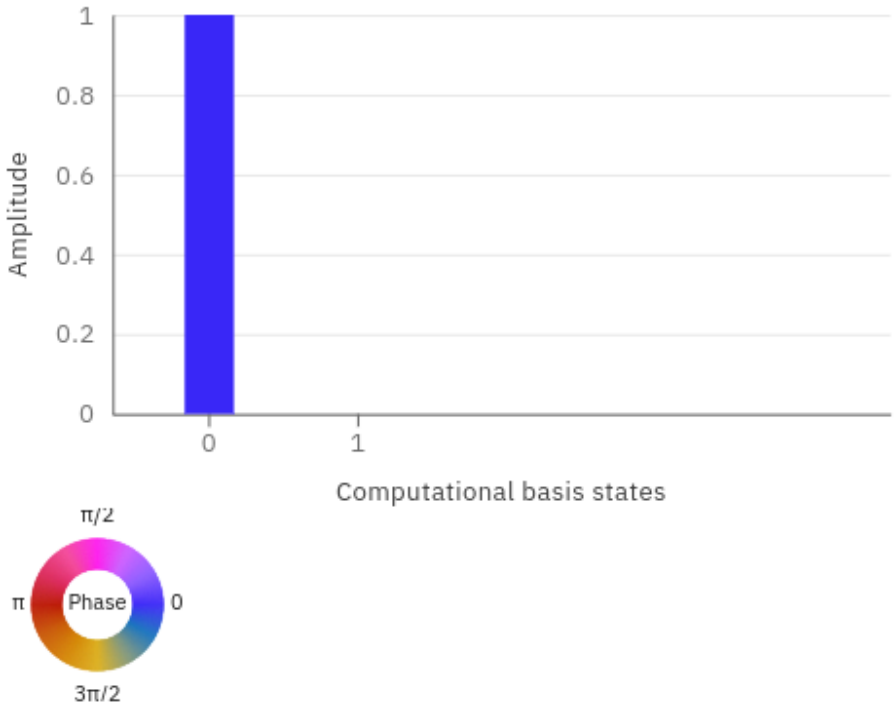


Figure 25: RHS = $|0\rangle$

$$HXH|0\rangle = HX(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = H(\frac{1}{\sqrt{2}}(|1\rangle + |0\rangle)) = \frac{1}{2}(|0\rangle - |1\rangle + |0\rangle + |1\rangle) = |0\rangle$$

Figure 26: LHS = HXH|1⟩

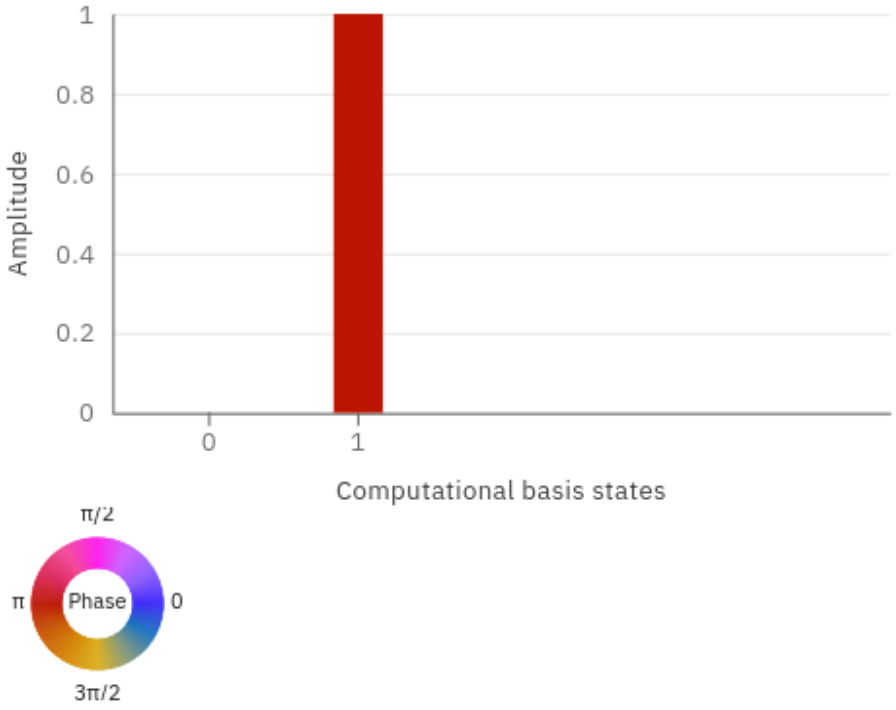


Figure 27: RHS = -|1⟩

$$HXH|1\rangle = HX(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) = H(\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)) = \frac{1}{2}(|0\rangle - |1\rangle - |0\rangle - |1\rangle) = -|1\rangle$$

Figure 28: LHS = Z|0⟩

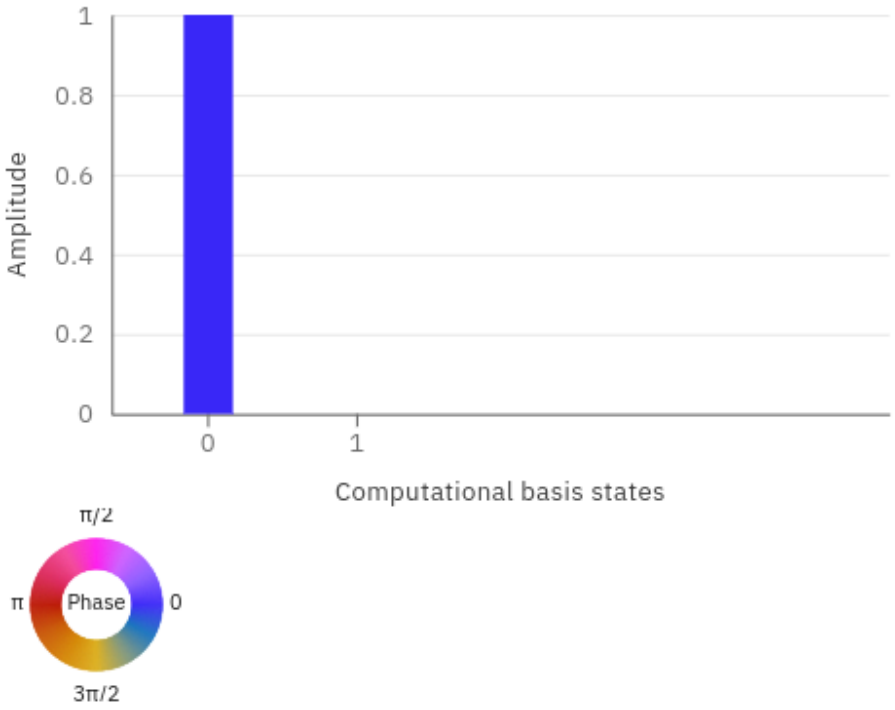


Figure 29: RHS = |0⟩

$Z|0\rangle = |0\rangle$

Figure 30: LHS = z|1⟩

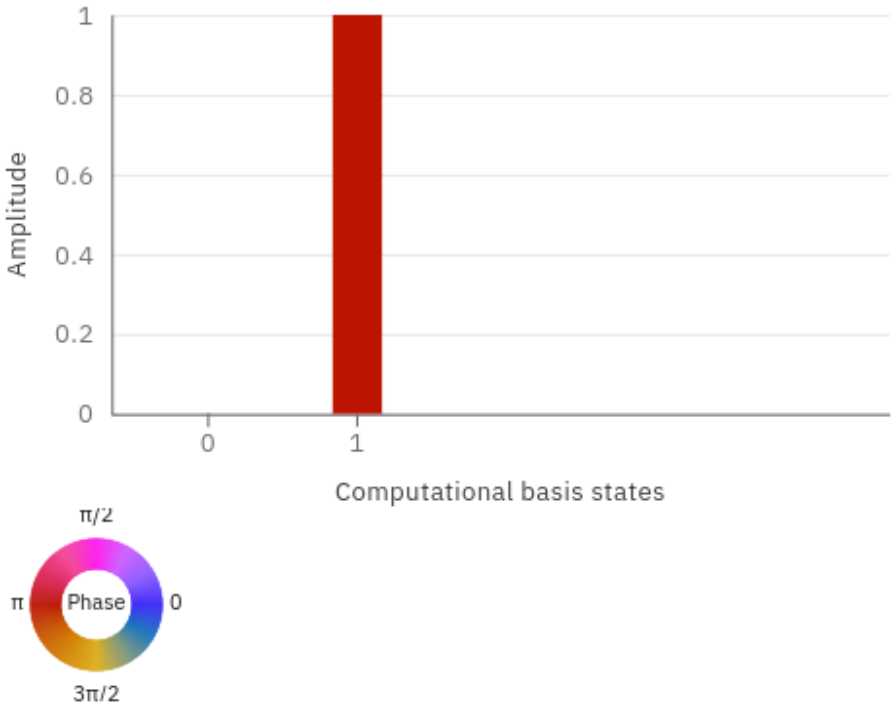


Figure 31: RHS = -|1⟩

$Z|1\rangle = -|1\rangle$

HXH $|0\rangle$ = Z $|0\rangle$ = $|0\rangle$ and HXH $|1\rangle$ = Z $|1\rangle$ = -$|1\rangle$

The identity has been verified and shown to be true.

(b) HYH = -Y

Figure 32: LHS = HYH$|0\rangle$

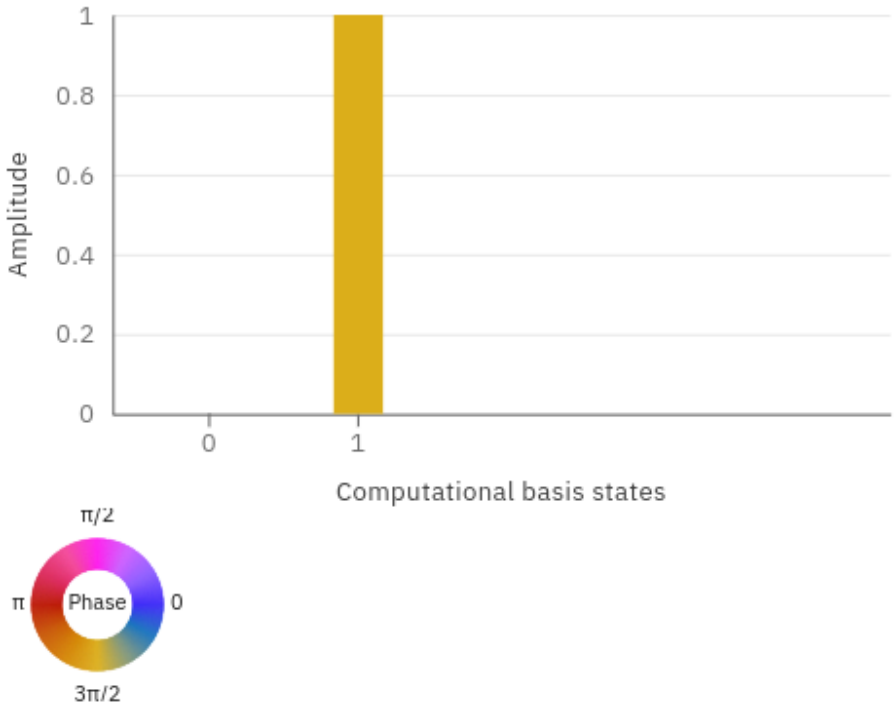


Figure 33: RHS = $-i\,|1\rangle$

$$\mathrm{HYH}|0\rangle = \mathrm{HX}(\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = \mathrm{H}(\tfrac{1}{\sqrt{2}}(\mathrm{i}\,|1\rangle - \mathrm{i}\,|0\rangle)) = \tfrac{1}{2}(\mathrm{i}\,|0\rangle - \mathrm{i}\,|1\rangle - \mathrm{i}\,|0\rangle - \mathrm{i}\,|1\rangle) = -\mathrm{i}\,|1\rangle$$

Figure 34: LHS $= \mathrm{HYH}|1\rangle$

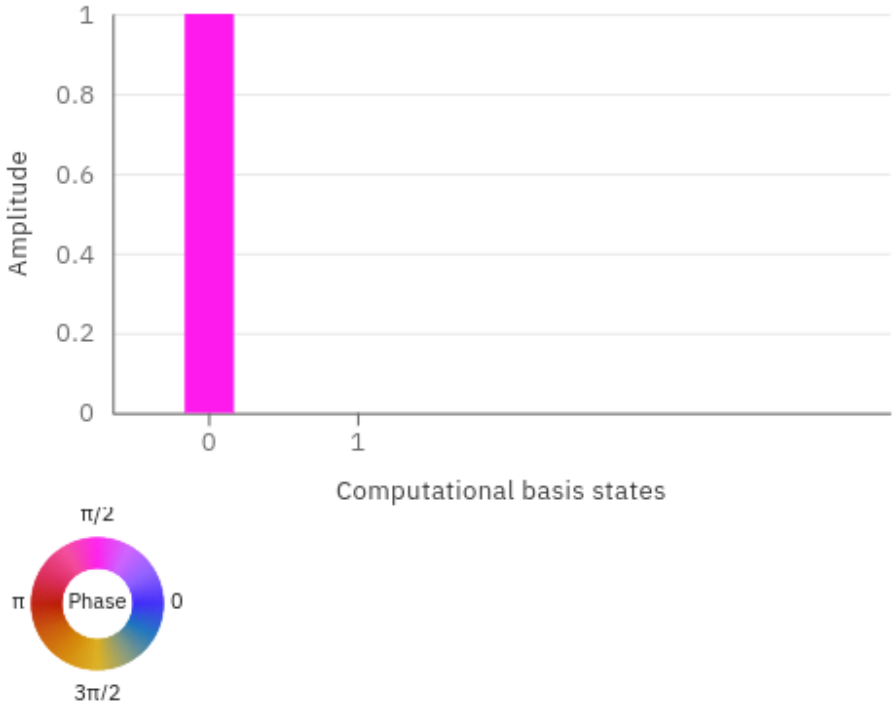


Figure 35: RHS $= i\,|0\rangle$

$$\mathrm{HYH}|1\rangle = \mathrm{HY}(\tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) = \mathrm{H}(\tfrac{1}{\sqrt{2}}(\mathrm{i}\,|1\rangle + \mathrm{i}\,|0\rangle)) = \tfrac{1}{2}(\mathrm{i}\,|0\rangle - \mathrm{i}\,|1\rangle + \mathrm{i}\,|0\rangle + \mathrm{i}\,|1\rangle) = \mathrm{i}\,|0\rangle$$

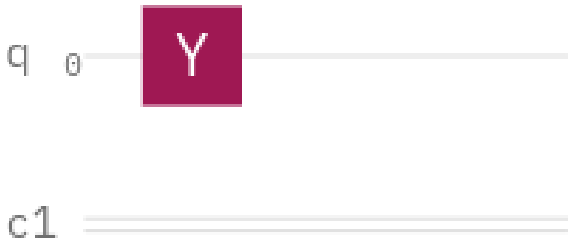


Figure 36: LHS $= \mathrm{Y}|0\rangle$

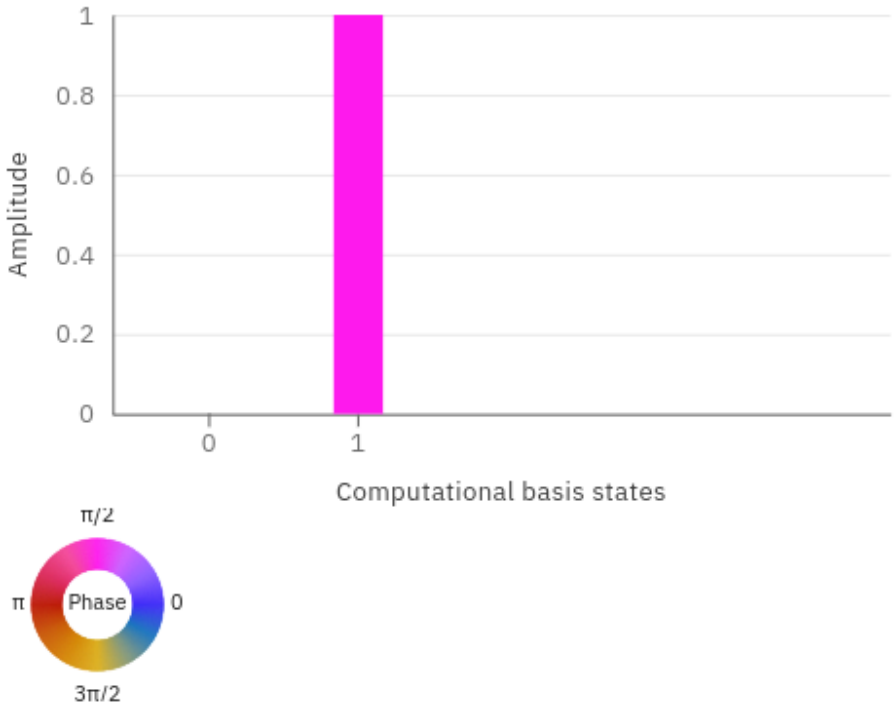


Figure 37: RHS $= i\,|1\rangle$

$\mathrm{Y}\,|0\rangle = \mathrm{i}\,|1\rangle$

Figure 38: LHS = Y|1⟩

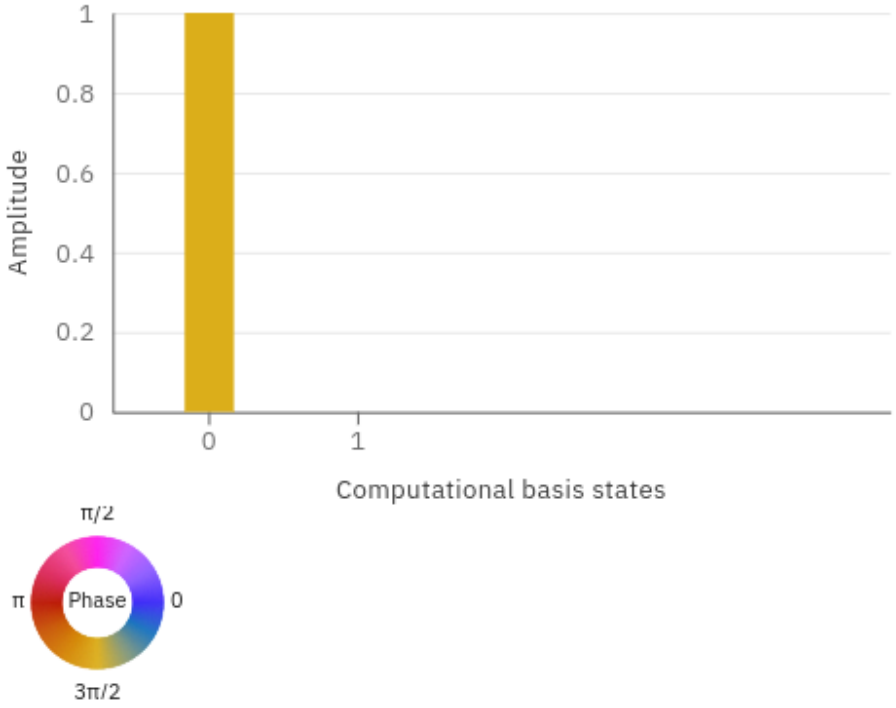


Figure 39: RHS $= -i\,|0\rangle$

$Y\,|1\rangle = -i\,|0\rangle$

HYH $|0\rangle$ = -Y $|0\rangle$ = $-i\,|1\rangle$ and HYH $|1\rangle$ = -Y $|1\rangle$ = $i\,|0\rangle$

The identity has been verified and shown to be true.

(c) HZH = X

Figure 40: LHS = HZH$|0\rangle$

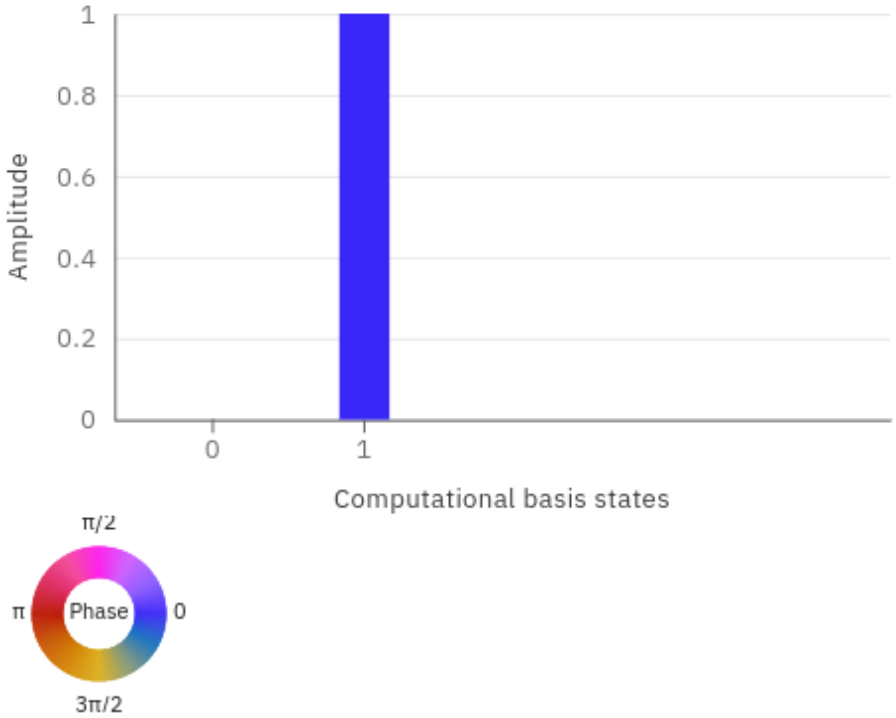


Figure 41: RHS = $|1\rangle$

$$HZH|0\rangle = HX(\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = H(\tfrac{1}{\sqrt{2}}(i|0\rangle - i|1\rangle)) = \tfrac{1}{2}(|0\rangle + |1\rangle - |0\rangle + |1\rangle) = |1\rangle$$

Figure 42: LHS $= HZH|1\rangle$

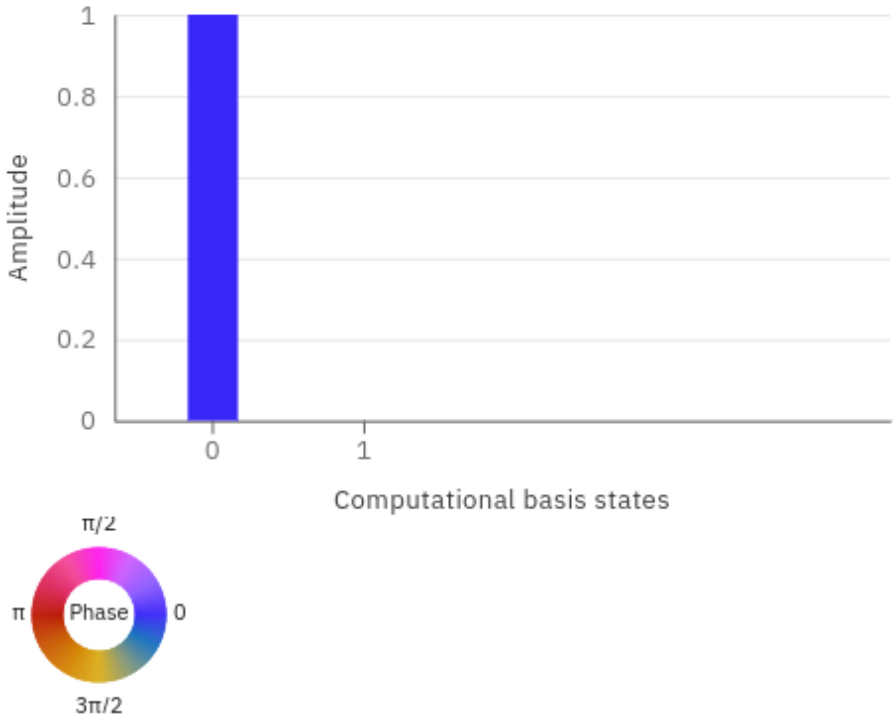


Figure 43: RHS $= |1\rangle$

$$HZH|1\rangle = HY(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) = H(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = \frac{1}{2}(|0\rangle + |1\rangle + |0\rangle - |1\rangle) = |0\rangle$$

Figure 44: LHS = X|0⟩

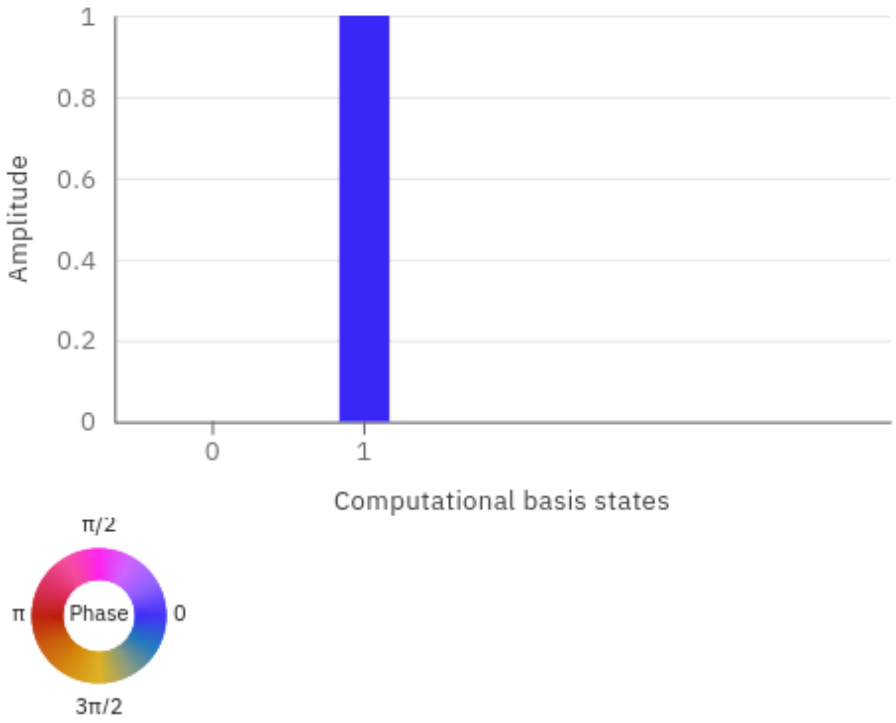


Figure 45: RHS = |1⟩

$Z|0\rangle = |1\rangle$

Figure 46: LHS = X|1⟩

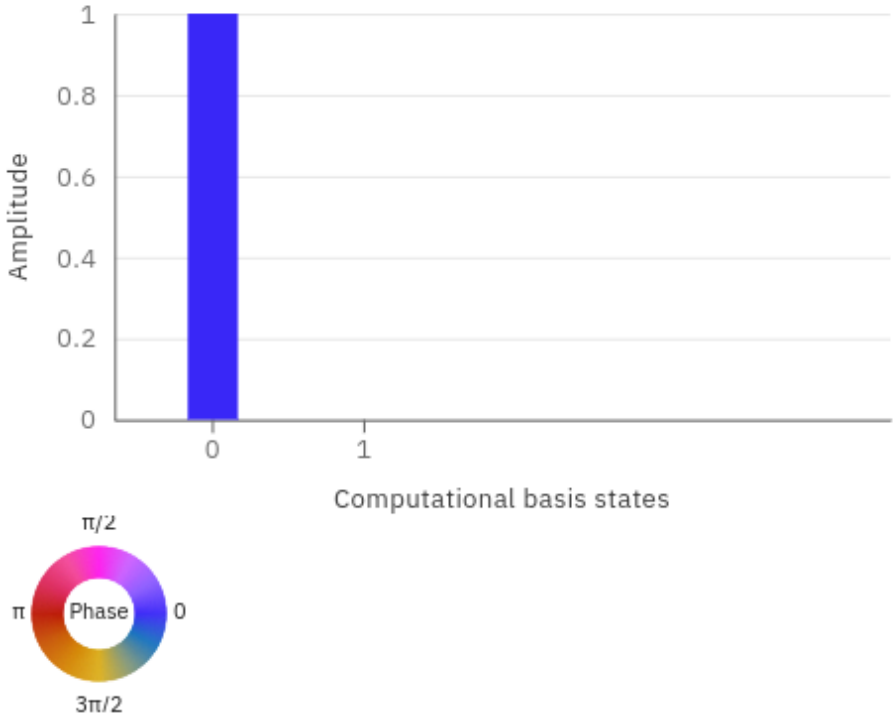


Figure 47: RHS =|0⟩

$X|1\rangle = |0\rangle$

$HZH|0\rangle = X|0\rangle = |1\rangle$ and $HZH|1\rangle = X|1\rangle = |1\rangle$

The identity has been verified and shown to be true.

(d) HTH = $R_x(\frac{\pi}{4})$

Figure 48: LHS = HTH$|0\rangle$

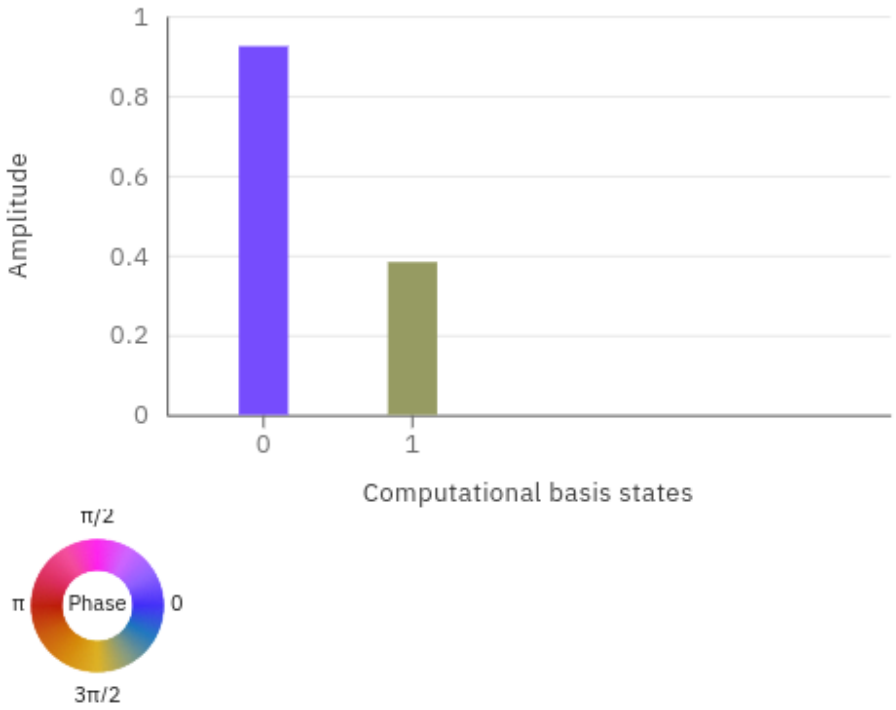


Figure 49: RHS = $|1\rangle$

Figure 50: LHS = HZH|1⟩

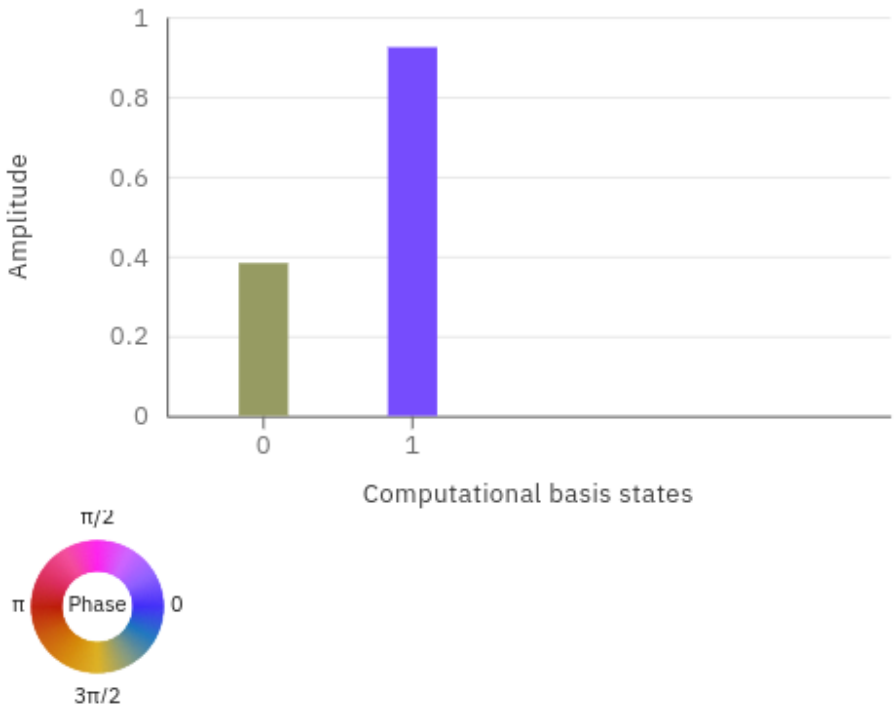


Figure 51: RHS

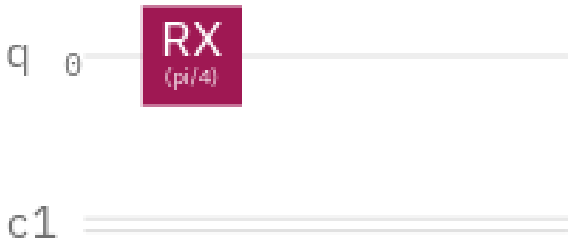


Figure 52: LHS = $R_X(\frac{pi}{4}|0\rangle$

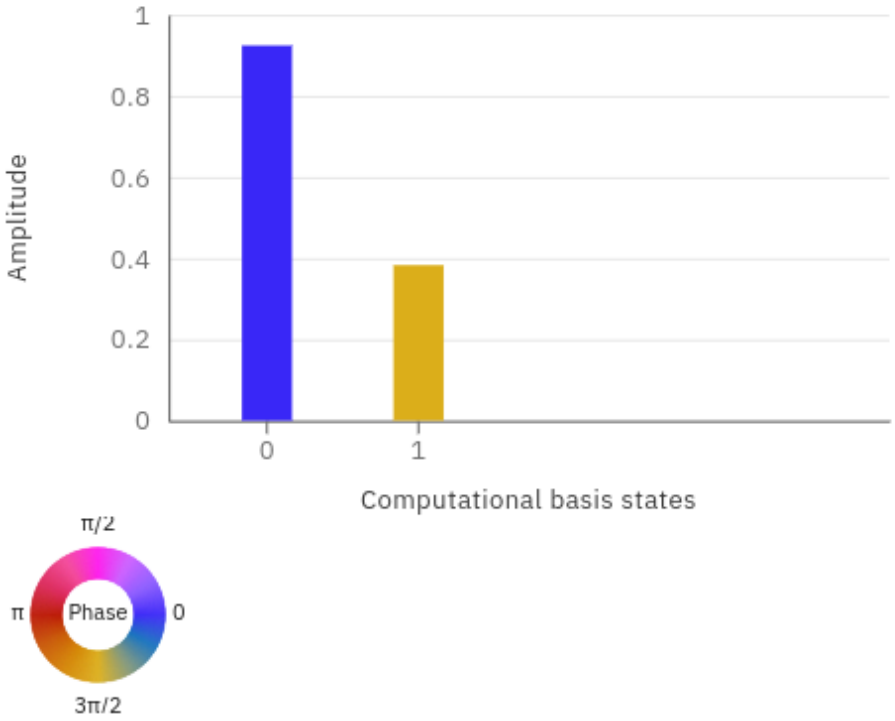


Figure 53: RHS

Figure 54: LHS = $R_X(\frac{pi}{4}|01\rangle$

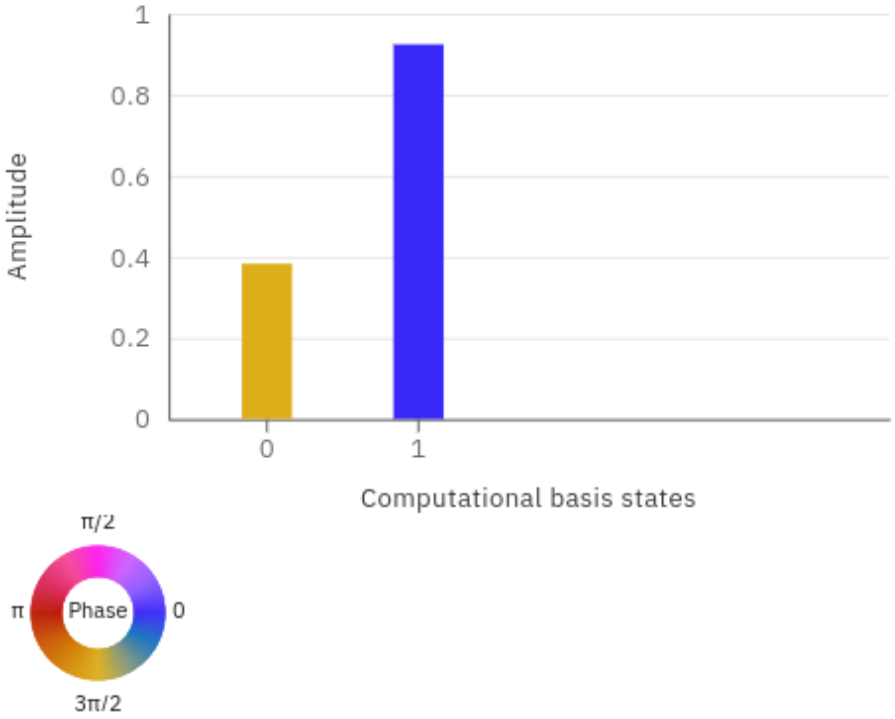


Figure 55: RHS

The identity has been verified and shown to be true.

3 Rabi Oscillations

(a) Solve the Rabi problem for $\hat{\sigma}_y$.

Schrödinger's equation:

$$i\hbar \frac{\partial |\psi\rangle}{\partial t} = H|\psi\rangle$$

We need to find $|\psi\rangle = \alpha|g\rangle + \beta|e\rangle$

where $|g\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $|e\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

given that at $t=0$, $|\psi(t)\rangle = |\psi(0)\rangle = |g\rangle$

The Hamiltonian for a driven qubit system is

$$\hat{H} = \frac{\hbar\omega_a}{2}\hat{\sigma}_z + \hbar\Omega\cos(\omega_d t + \phi_0)\hat{\sigma}_y$$

$$\hat{H} = \hat{H}_0 + \hat{H}_I$$

$\hat{H}_0 = \frac{\hbar\omega_a}{2}\hat{\sigma}_z$ is the bare/uncoupled Hamiltonian

$\hat{H}_I = \hbar\Omega\cos(\omega_d t + \phi_0)\hat{\sigma}_y$ is the interaction (drive) Hamiltonian

where Ω is the Rabi frequency ($\Omega \propto gV_0$)

To solve the Rabi problem, we shall go into the interaction picture as follows:

let $|\psi_I\rangle = e^{\frac{iH_0 t}{\hbar}}|\psi\rangle$ — ①

such that $|\psi_I(t)\rangle = C_g(t)|g\rangle + C_e(t)|e\rangle$

and Schrödinger equation is

$$i\hbar \frac{\partial |\psi_I\rangle}{\partial t} = \tilde{H}|\psi_I\rangle \quad \text{in interaction picture} \quad — ②$$

Also at $t=0$, $|\psi_I(t)\rangle = |\psi_I(0)\rangle = |\psi(0)\rangle = |g\rangle$.

Substituting for $|\psi_I\rangle$ from ① in ②

$$i\hbar \frac{\partial}{\partial t}\left(e^{iH_0 t/\hbar}|\psi\rangle\right) = \tilde{H}\left(e^{iH_0 t/\hbar}|\psi\rangle\right)$$

$$i\hbar\left\{\frac{iH_0}{\hbar}e^{iH_0 t/\hbar}|\psi\rangle + e^{iH_0 t/\hbar}\frac{\partial|\psi\rangle}{\partial t}\right\} = \tilde{H}e^{iH_0 t/\hbar}|\psi\rangle$$

$$-\hat{H}_0 e^{\frac{i\hat{H}_0 t}{\hbar}}|\psi\rangle + i\hbar e^{i\hat{H}_0 t/\hbar}\frac{\partial|\psi\rangle}{\partial t} = \hat{H} e^{i\hat{H}_0 t/\hbar}|\psi\rangle \quad -(3)$$

Pre multiplying $e^{-i\hat{H}_0 t/\hbar}$ to (3)

$$-e^{-i\hat{H}_0 t/\hbar}\hat{H}_0 e^{i\hat{H}_0 t/\hbar}|\psi\rangle + i\hbar e^{\frac{-i\hat{H}_0 t}{\hbar}} e^{\frac{i\hat{H}_0 t}{\hbar}}\frac{\partial|\psi\rangle}{\partial t} = e^{\frac{-i\hat{H}_0 t}{\hbar}}\tilde{H} e^{\frac{i\hat{H}_0 t}{\hbar}}|\psi\rangle$$

$e^{-i\hat{H}_0 t/\hbar}$ and H_0 commute and $e^{-i\hat{H}_0 t/\hbar} e^{i\hat{H}_0 t/\hbar} = I$

$$\therefore -\hat{H}_0|\psi\rangle + i\hbar\frac{\partial|\psi\rangle}{\partial t} = e^{\frac{-i\hat{H}_0 t}{\hbar}}\tilde{H} e^{\frac{i\hat{H}_0 t}{\hbar}}|\psi\rangle$$

But $i\hbar\frac{\partial|\psi\rangle}{\partial t} = \hat{H}|\psi\rangle = (\hat{H}_0 + \hat{H}_I)|\psi\rangle$

$$\therefore -\hat{H}_0|\psi\rangle + \hat{H}_0|\psi\rangle + \hat{H}_I|\psi\rangle = e^{-i\hat{H}_0 t/\hbar}\tilde{H} e^{i\hat{H}_0 t/\hbar}|\psi\rangle$$

$$\therefore \hat{H}_I = e^{-i\hat{H}_0 t/\hbar}\tilde{H} e^{i\hat{H}_0 t/\hbar}$$

Let $e^{i\hat{H}_0 t/\hbar} = U$ and $e^{-i\hat{H}_0 t/\hbar} = U^{\dagger} = U^{-1}$

$$\tilde{H} = U H_I U^{-1} \quad \text{(similarity transformation)}$$

$$\tilde{H} = e^{i\hat{H}_0 t/\hbar} H_I e^{-i\hat{H}_0 t/\hbar}$$

This is the Hamiltonian in the interaction picture

Since $H_0 = \frac{\hbar\omega_a}{2}\hat{\sigma}_z$

$$U = e^{i\hat{H}_0 t/\hbar} = e^{\frac{i\hbar\omega_a t\hat{\sigma}_z}{2\hbar}} = e^{\frac{i\omega_a t\hat{\sigma}_z}{2}}$$

and $U^{-1} = e^{-i\hat{H}_0 t/\hbar} = e^{-\frac{i\omega_a t\hat{\sigma}_z}{2}}$

and $H_I = \hbar\Omega\cos(\omega_d t + \phi_0)\hat{\sigma}_y$

$$\therefore \tilde{H} = e^{\frac{i\omega_a t}{2}\hat{\sigma}_z}\hbar\Omega\cos(\omega_d t + \phi_0)\hat{\sigma}_y e^{-\frac{i\omega_a t\hat{\sigma}_z}{2}}$$

$$\tilde{H} = \hbar\Omega\cos(\omega_d t + \phi_0)\, e^{\frac{i\omega_a t}{2}\hat{\sigma}_z}\cdot\hat{\sigma}_y\cdot e^{-\frac{i\omega_a t}{2}\hat{\sigma}_z}$$

$\sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = -i\sigma_+ + i\sigma_-$ where $\sigma_+ = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $\sigma_- = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$

$$\therefore \tilde{H} = \hbar\Omega\cos(\omega_d t + \phi_0)\, e^{\frac{i\omega_a t}{2}\hat{\sigma}_z}(-i\sigma_+ + i\sigma_-)\, e^{-\frac{i\omega_a t}{2}\hat{\sigma}_z}$$

Using Baker Campbell Hausdorf Formula, we have

$$e^{\frac{i\omega_a t}{2}\hat{\sigma}_2}\,\sigma_+\,e^{\frac{-i\omega_a t\hat{\sigma}_2}{2}} = \sigma_+ e^{i\omega_a t}$$

$$e^{\frac{i\omega_a t\hat{\sigma}_2}{2}}\,\sigma_-\,e^{\frac{-i\omega_a t\hat{\sigma}_2}{2}} = \sigma_- e^{-i\omega_a t}$$

$$\therefore \tilde{H} = \hbar\Omega\left(\frac{e^{i(\omega_d t+\phi_0)} + e^{-i(\omega_d t+\phi_0)}}{2}\right)\left(-i\sigma_+ e^{i\omega_a t} + i\sigma_- e^{-i\omega_a t}\right)$$

$$\tilde{H} = \frac{\hbar\Omega}{2}\left[\begin{array}{ll} -i\sigma_+ e^{i((\omega_a-\omega_d)t-\phi_0)} & +i\sigma_- e^{-i[(\omega_a-\omega_d)t-\phi_0]} \\ -i\sigma_+ e^{i[(\omega_a+\omega_d)t+\phi_0]} & +i\sigma_- e^{-i[(\omega_a+\omega_d)t+\phi_0]}\end{array}\right]$$

If $\omega_a - \omega_d \ll \omega_a$ or ω_d, then ($\omega_a \approx \omega_d$ usually) the first two terms with $(\omega_a-\omega_a)$ rotate slowly and the last two terms with $(\omega_a+\omega_d)$ rotate rapidly.

$\therefore$ We can ignore the $(\omega_a+\omega_d)$ terms, to get

$$\boxed{\tilde{H} = \frac{\hbar\Omega}{2}\left(-i\sigma_+ e^{i[(\omega_a-\omega_d)t-\phi_0]} + i\sigma_- e^{-i[(\omega_a-\omega_d)t-\phi_0]}\right)}$$

This is the hamiltonian obtained after applying the Rotating wave Approximation.

The Schrödinger equation becomes.

$$i\hbar\frac{\partial|\psi_I\rangle}{\partial t} = \frac{\hbar\Omega}{2}\left[-i\sigma_+ e^{i[(\omega_a-\omega_d)t-\phi_0]} + i\sigma_- e^{-i[(\omega_a-\omega_d)t-\phi_0]}\right]|\psi\rangle_I.$$

Substitute $|\psi_I\rangle = C_g|g\rangle + C_e|e\rangle$ in this equation.

$$i\hbar\frac{\partial}{\partial t}(C_g|g\rangle + C_e|e\rangle) = \frac{\hbar\Omega}{2}\left(-i\sigma_+ e^{i[(\omega_a-\omega_d)t-\phi_0]} + i\sigma_- e^{-i[(\omega_a-\omega_d)t-\phi_0]}\right) \times (C_g|g\rangle + C_e|e\rangle)$$

We know that, $\sigma_+|g\rangle = |e\rangle$, $\sigma_-|e\rangle = |g\rangle$

$\sigma_+|e\rangle = |0\rangle$, $\sigma_-|g\rangle = |0\rangle$.

$$\therefore i\hbar(\dot{C}_g|g\rangle + \dot{C}_e|e\rangle) = \frac{\hbar\Omega}{2}\left[-iC_g e^{i[(\omega_a-\omega_d)t-\phi_0]}|e\rangle + iC_e e^{-i[(\omega_a-\omega_d)t-\phi_0]}|g\rangle\right]$$

$|g\rangle$ and $|e\rangle$ are orthogonal, $\langle g|e\rangle = 0$.
So we can equate their coefficient terms.
Equating coefficients of $|g\rangle$, we get

$$i\hbar \dot{C}_g = \frac{\hbar\Omega}{2} i e^{-i[(\omega_a-\omega_d)t-\phi_0]} C_e$$

$$\dot{C}_g = \frac{\Omega}{2} e^{-i[(\omega_a-\omega_d)t-\phi_0]} C_e \quad \text{—— (4)}$$

Equating coefficients of $|e\rangle$, we get

$$i\hbar \dot{C}_e = \frac{\hbar\Omega}{2}\left(-i C_g e^{i[(\omega_a-\omega_d)t-\phi_0]}\right)$$

$$\dot{C}_e = -\frac{\Omega}{2} e^{i[(\omega_a-\omega_d)t-\phi_0]} C_g \quad \text{—— (5)}$$

Differentiate (5) with respect to time to get

$$\ddot{C}_e = -\frac{\Omega}{2}\left[\dot{C}_g e^{i[(\omega_a-\omega_d)t-\phi_0]} + C_g\, i(\omega_a-\omega_d)\, e^{i[(\omega_a-\omega_d)t-\phi_0]}\right]$$

Substituting values of $\dot{C}_g$ and C_g from (4) and (5) respectively, we get

$$\ddot{C}_e = -\frac{\Omega}{2}\left[\frac{\Omega}{2} e^{-i[(\omega_a-\omega_d)t-\phi_0]} e^{i[(\omega_a-\omega_d)t-\phi_0]} C_e + \left(-\frac{2}{\Omega}\dot{C}_e\right) i(\omega_a-\omega_d)\right]$$

$$\ddot{C}_e = -\frac{\Omega^2}{4} C_e + i(\omega_a-\omega_d)\dot{C}_e$$

The differential equation in C_e is

$$\ddot{C}_e + i(\omega_d-\omega_a)\dot{C}_e + \frac{\Omega^2}{4} C_e = 0$$

The initial conditions are, at $t=0$.

$C_e(t)|_{t=0} = C_e(0) = 0$ and $C_g(t)|_{t=0} = C_g(0) = 1$

Setting $t=0$ in equation (5),

$$\dot{C}_e(0) = -\frac{\Omega}{2} e^{-i\phi_0}$$

Solving the characteristic equation,

$$\lambda^2 + i(\omega_d - \omega_a)\lambda + \frac{\Omega^2}{4} = 0$$

$$\lambda_{\pm} = \frac{-i(\omega_d - \omega_a) \pm \sqrt{-(\omega_d - \omega_a)^2 - \Omega^2}}{2}$$

Let $(\omega_d - \omega_a) = \Delta$

$$\lambda_+ = i\left(\frac{-\Delta + \sqrt{\Delta^2 + \Omega^2}}{2}\right) \quad \text{and} \quad \lambda_- = i\left(\frac{-\Delta - \sqrt{\Delta^2 + \Omega^2}}{2}\right)$$

$$C_e(t) = A e^{\lambda_+ t} + B e^{\lambda_- t}$$

At $t = 0$, $\quad C_e(0) = A + B = 0$

$\therefore A = -B$

and, $\quad \dot{C}_e(t) = A\lambda_+ e^{\lambda_+ t} + B\lambda_- e^{\lambda_- t}$

at $t = 0$, $\quad \dot{C}_e(0) = \lambda_+ A + \lambda_- B = -\frac{\Omega}{2} e^{-i\phi_0}$

Since $B = -A$, $\quad (\lambda_+ - \lambda_-) A = -\frac{\Omega}{2} e^{-i\phi_0}$

$$(\lambda_+ - \lambda_-) = i\sqrt{\Delta^2 + \Omega^2}$$

$$\therefore A = +\frac{i\Omega e^{-i\phi_0}}{2\sqrt{\Delta^2 + \Omega^2}} \quad \text{and} \quad B = -\frac{i\Omega e^{-i\phi_0}}{2\sqrt{\Delta^2 + \Omega^2}}$$

$$\therefore C_e(t) = \frac{i\Omega e^{-i\phi_0}}{2\sqrt{\Delta^2 + \Omega^2}} e^{i\left(\frac{-\Delta + \sqrt{\Delta^2 + \Omega^2}}{2}\right)t} - \frac{i\Omega e^{-i\phi_0}}{2\sqrt{\Delta^2 + \Omega^2}} e^{i\left(\frac{-\Delta - \sqrt{\Delta^2 + \Omega^2}}{2}\right)t}$$

$$C_e(t) = \frac{i\Omega e^{-i\phi_0} e^{-i\Delta}}{2\sqrt{\Delta^2 + \Omega^2}} \left(e^{\frac{i\sqrt{\Delta^2 + \Omega^2}\, t}{2}} - e^{-\frac{i\sqrt{\Delta^2 + \Omega^2}\, t}{2}} \right)$$

$$C_e(t) = -\frac{\Omega e^{-i\phi_0} e^{-i\Delta}}{\sqrt{\Delta^2 + \Omega^2}} \sin\left(\frac{\sqrt{\Delta^2 + \Omega^2}}{2} t\right)$$

Let $\Omega_R = \sqrt{\Delta^2 + \Omega^2}$

Ω_R is called the generalised Rabi frequency.

$$C_e(t) = \frac{-\Omega}{\Omega_R} e^{-i(\Delta+\phi_0)} \sin\left(\frac{\Omega_R t}{2}\right)$$

Similarly, we can solve for $C_g(t)$ using the differential equation formed using ④ and ⑤

$$\ddot{C}_g + i(\omega_a - \omega_d)\dot{C}_g + \frac{\Omega^2}{4} C_g = 0$$

The characteristic equation $\lambda^2 + i(\omega_a - \omega_d)\lambda + \frac{\Omega^2}{4} = 0$

has the roots $\lambda_{\pm} = i\left(\frac{\Delta \pm \sqrt{\Delta^2+\Omega^2}}{2}\right)$

and the initial conditions are $C_g(0) = 1$ and $\dot{C}_g(0) = 0$

On solving, we get

$$C_g(t) = \frac{-\Delta + \sqrt{\Delta^2+\Omega^2}}{2\sqrt{\Delta^2+\Omega^2}} e^{i\left(\frac{\Delta+\sqrt{\Delta^2+\Omega^2}}{2}\right)t} + \frac{\Delta+\sqrt{\Delta^2+\Omega^2}}{2\sqrt{\Delta^2+\Omega^2}} e^{i\left(\frac{\Delta-\sqrt{\Delta^2+\Omega^2}}{2}\right)t}$$

$$C_g(t) = \frac{(-\Delta+\Omega_R)}{2\Omega_R} e^{i\left(\frac{\Delta+\Omega_R}{2}\right)t} + \frac{(\Delta+\Omega_R)}{2\Omega_R} e^{i\left(\frac{\Delta-\Omega_R}{2}\right)t}.$$

On resonance, $\omega_a = \omega_d$,

therefore $\Delta = 0$ and $\Omega_R = \Omega$

$$C_e(t) = -e^{-i\phi_0} \sin\left(\frac{\Omega t}{2}\right)$$

and $$C_g(t) = \frac{e^{\frac{i\Omega t}{2}} + e^{-\frac{i\Omega t}{2}}}{2}$$

$$C_g(t) = \cos\left(\frac{\Omega t}{2}\right)$$

Let $\theta = \Omega_R t$

$$|\psi_I(t)\rangle = C_g(t)|g\rangle + C_e(t)|e\rangle$$

$$|\psi_I(t)\rangle = \cos\left(\frac{\theta}{2}\right)|g\rangle - e^{-i\phi_0}\sin\left(\frac{\theta}{2}\right)|e\rangle$$

We have $|\psi_I\rangle = e^{iH_0t/\hbar}|\psi\rangle$

$\therefore |\psi\rangle = e^{-iH_0t/\hbar}|\psi_I\rangle$

$|\psi\rangle = e^{-iH_0t/\hbar}\left(\cos(\theta/2)|g\rangle - e^{-i\phi_0}\sin(\theta/2)|e\rangle\right)$

We have

$e^{-iH_0t/\hbar}|g\rangle = e^{\frac{i\omega_a t}{2}}|g\rangle$

and

$e^{-iH_0t/\hbar}|e\rangle = -e^{-\frac{i\omega_a t}{2}}|e\rangle$

$|\psi\rangle = \cos(\theta/2)\, e^{\frac{i\omega_a t}{2}}|g\rangle + e^{-i\phi_0}\sin(\theta/2)\, e^{-\frac{i\omega_a t}{2}}|e\rangle$

Taking out the global phase factor of $e^{\frac{i\omega_a t}{2}}$,

$$\boxed{|\psi\rangle = \cos(\theta/2)|g\rangle + e^{-i\phi_0}\sin(\theta/2)\, e^{-i\omega_a t}|e\rangle}$$

This is the general solution of the Rabi problem with $\theta \simeq \Omega_R t$ where $\Omega_R = \sqrt{\Omega^2 + \Delta^2}$

Ω_R is the generalised or detuned Rabi frequency

Question 3 (b)

Finding the Transition frequency of Transmon

[1]:
```
from qiskit.tools.jupyter import *
```

[2]:
```
from qiskit import IBMQ
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q', group='open', project='main')
backend = provider.get_backend('ibmq_armonk')
```

We verify that the backend supports Pulse features by checking the backend configuration.

[3]:
```
backend_config = backend.configuration()
assert backend_config.open_pulse, "Backend doesn't support Pulse"
```

We can find the sampling time for the backend pulses within the backend configuration.

[4]:
```
dt = backend_config.dt
print(f"Sampling time: {dt*1e9} ns")    # The configuration returns dt in␣
 ↪seconds, so multiply by
                                        # 1e9 to get nanoseconds
```

```
Sampling time: 0.2222222222222222 ns
```

The backend defaults provide a starting point for how to use the backend. It contains estimates for qubit frequencies and default programs to enact basic quantum operators. We can access them with the following:

[6]:
```
backend_defaults = backend.defaults()
```

1. Finding the qubit Frequency using a Frequency Sweep

The qubit frequency is the difference in energy between the ground and excited states, which we label the $|0\rangle$ and $|1\rangle$ states, respectively.

First, we define the frequency range that will be swept in search of the qubit. Since this can be arbitrarily broad, we restrict ourselves to a window of 40 MHz around the estimated qubit frequency in `backend_defaults`. We step the frequency in units of 1 MHz.

[7]:
```
import numpy as np

# unit conversion factors -> all backend properties returned in SI (Hz, sec,␣
↪etc.)
GHz = 1.0e9 # Gigahertz
MHz = 1.0e6 # Megahertz
us = 1.0e-6 # Microseconds
ns = 1.0e-9 # Nanoseconds

# We will find the qubit frequency for the following qubit.
qubit = 0
# We will define memory slot channel 0.
mem_slot = 0

# The sweep will be centered around the estimated qubit frequency.
center_frequency_Hz = backend_defaults.qubit_freq_est[qubit]        # The␣
↪default frequency is given in Hz
                                                                    # warning:␣
↪this will change in a future release
print(f"Qubit {qubit} has an estimated frequency of {center_frequency_Hz / GHz}␣
↪GHz.")

# scale factor to remove factors of 10 from the data
scale_factor = 1e-14

# We will sweep 40 MHz around the estimated frequency
frequency_span_Hz = 40 * MHz
# in steps of 1 MHz.
frequency_step_Hz = 1 * MHz

# We will sweep 20 MHz above and 20 MHz below the estimated frequency
frequency_min = center_frequency_Hz - frequency_span_Hz / 2
frequency_max = center_frequency_Hz + frequency_span_Hz / 2
# Construct an np array of the frequencies for our experiment
frequencies_GHz = np.arange(frequency_min / GHz,
                            frequency_max / GHz,
                            frequency_step_Hz / GHz)

print(f"The sweep will go from {frequency_min / GHz} GHz to {frequency_max /␣
↪GHz} GHz \
in steps of {frequency_step_Hz / MHz} MHz.")
```

```
Qubit 0 has an estimated frequency of 4.971613487748122 GHz.
The sweep will go from 4.951613487748122 GHz to 4.991613487748122 GHz in steps
of 1.0 MHz.
```

We will create a pulse schedule by defining this frequency as a parameter using the parameter class. First, we will set the required values duration, sigma, and channel.

Then we will set the pulse flow so that the specified pulses are executed sequentially. We will define the pulse frequency, the pulse used in the experiment, and the measurement pulse. Here, the pulse used in the experiment specifies the drive pulse, which is a Gaussian pulse.

At each frequency, we will send a drive pulse of that frequency to the qubit and measure immediately after the pulse.

[8]:
```
# samples need to be multiples of 16
def get_closest_multiple_of_16(num):
    return int(num + 8 ) - (int(num + 8 ) % 16)
```

[9]:
```
# Convert seconds to dt
def get_dt_from(sec):
    return get_closest_multiple_of_16(sec/dt)
```

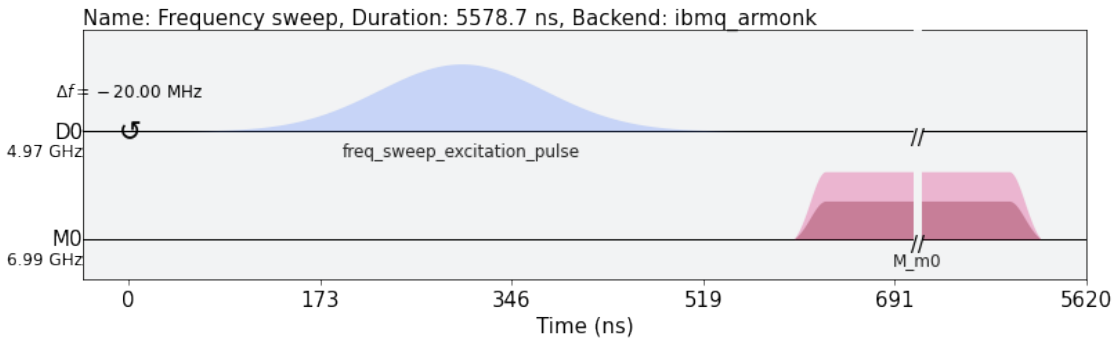
[10]:
```
from qiskit import pulse                    # This is where we access all of our␣
 ↪Pulse features!
from qiskit.circuit import Parameter       # This is Parameter Class for␣
 ↪variable parameters.


# Drive pulse parameters (us = microseconds)
drive_sigma_sec = 0.075 * us                              # This determines the␣
 ↪actual width of the gaussian
drive_duration_sec = drive_sigma_sec * 8                  # This is a truncating␣
 ↪parameter, because gaussians don't have
                                                          # a natural finite␣
 ↪length
drive_amp = 0.05


# Create the base schedule
# Start with drive pulse acting on the drive channel
freq = Parameter('freq')
with pulse.build(backend=backend, default_alignment='sequential',␣
 ↪name='Frequency sweep') as sweep_sched:
    drive_duration = get_closest_multiple_of_16(pulse.
 ↪seconds_to_samples(drive_duration_sec))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.set_frequency(freq, drive_chan)
    # Drive pulse samples
    pulse.play(pulse.Gaussian(duration=drive_duration,
                              sigma=drive_sigma,
                              amp=drive_amp,
                              name='freq_sweep_excitation_pulse'), drive_chan)
    # Define our measurement pulse
```

```
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])


# Create the frequency settings for the sweep (MUST BE IN HZ)
frequencies_Hz = frequencies_GHz*GHz
schedules = [sweep_sched.assign_parameters({freq: f}, inplace=False) for f in␣
 ↪frequencies_Hz]
```

As a sanity check, it's always a good idea to look at the pulse schedule. This is done using `schedule.draw()` as shown below.

[11]:
```
schedules[0].draw(backend=backend)
```

[11]:



We request that each schedule (each point in our frequency sweep) is repeated num_shots_per_frequency times in order to get a good estimate of the qubit response.

We also specify measurement settings. meas_level=0 returns raw data (an array of complex values per shot), meas_level=1 returns kerneled data (one complex value per shot), and meas_level=2 returns classified data (a 0 or 1 bit per shot). We choose meas_level=1 to replicate what we would be working with if we were in the lab, and hadn't yet calibrated the discriminator to classify 0s and 1s. We ask for the 'avg' of the results, rather than each shot individually.

[12]:
```
num_shots_per_frequency = 1024

job = backend.run(schedules,
                  meas_level=1,
                  meas_return='avg',
                  shots=num_shots_per_frequency)
```

It is always a good idea to monitor the job status by using job_monitor()

[13]:
```
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

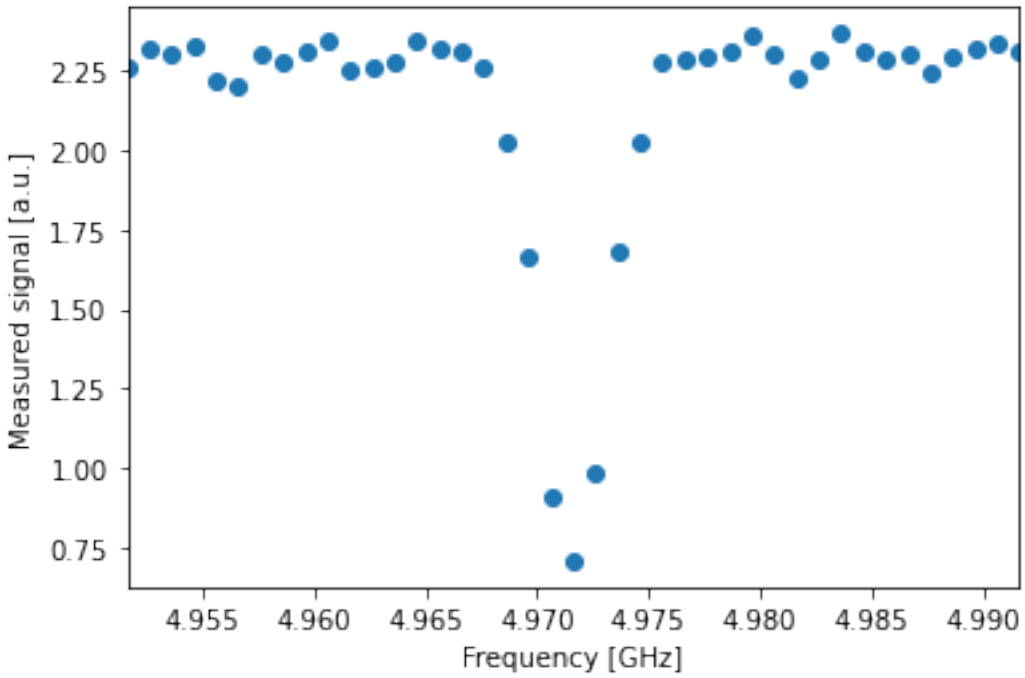
Job Status: job has successfully run

Once the job is run, the results can be retrieved using:

[14]:
```
frequency_sweep_results = job.result(timeout=120) # timeout parameter set to␣
↪120 seconds
```

Extraction and plotting of the obtained results:

[15]:
```
import matplotlib.pyplot as plt

sweep_values = []
for i in range(len(frequency_sweep_results.results)):
    # Get the results from the ith experiment
    res = frequency_sweep_results.get_memory(i)*scale_factor
    # Get the results for `qubit` from this experiment
    sweep_values.append(res[qubit])

plt.scatter(frequencies_GHz, np.real(sweep_values)) # plot real part of sweep␣
↪values
plt.xlim([min(frequencies_GHz), max(frequencies_GHz)])
plt.xlabel("Frequency [GHz]")
plt.ylabel("Measured signal [a.u.]")
plt.show()
```



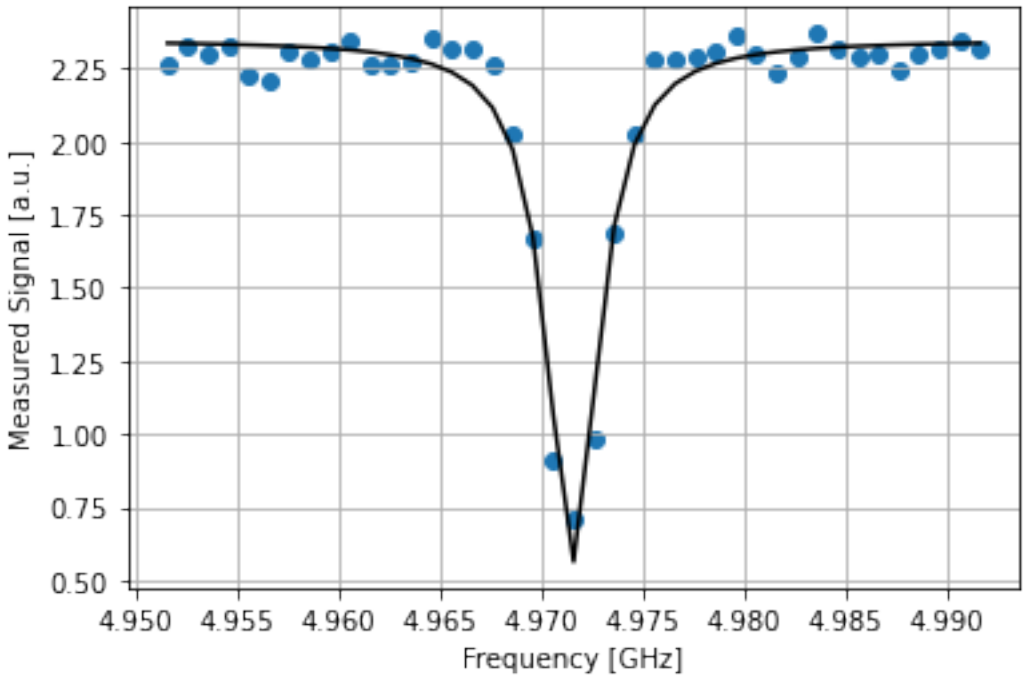
As you can see above, the peak near the center corresponds to the location of

the qubit frequency. The signal shows power-broadening, which is a signature that we are able to drive the qubit off-resonance as we get close to the center frequency. To get the value of the peak frequency, we will fit the values to a resonance response curve, which is typically a Lorentzian shape.

[16]:
```
from scipy.optimize import curve_fit

def fit_function(x_values, y_values, function, init_params):
    fitparams, conv = curve_fit(function, x_values, y_values, init_params)
    y_fit = function(x_values, *fitparams)

    return fitparams, y_fit
```

[23]:
```
fit_params, y_fit = fit_function(frequencies_GHz,
                                 np.real(sweep_values),
                                 lambda x, A, q_freq, B, C: (A / np.pi) * (B /␣
↪((x - q_freq)**2 + B**2)) + C,
                                 [-5, 4.975, 1, 1] # initial parameters for␣
↪curve_fit
                                )
```

[26]:
```
plt.scatter(frequencies_GHz, np.real(sweep_values))
plt.plot(frequencies_GHz, y_fit,color='black')
plt.xlabel("Frequency [GHz]")
plt.ylabel("Measured Signal [a.u.]")
plt.grid(True)
plt.show()
```



[27]:
```
A, rough_qubit_frequency, B, C = fit_params
rough_qubit_frequency = rough_qubit_frequency*GHz # make sure qubit freq is in␣
↪Hz
print(f"We've updated our qubit frequency estimate from "
      f"{round(backend_defaults.qubit_freq_est[qubit] / GHz, 5)} GHz to␣
↪{round(rough_qubit_frequency/GHz, 5)} GHz.")
```

```
We've updated our qubit frequency estimate from 4.97161 GHz to 4.97157 GHz.
```

2. Using Rabi Oscillations (Sweeping Amplitude)

The next step is to determine the strength of a π pulse. This is also called the X or $X180$ gate, or bit-flip operator for a two level system.

We will change the drive amplitude in small increments and measuring the state of the qubit each time. We expect to see oscillations which are commonly named Rabi oscillations, as the qubit goes from $|0\rangle$ to $|1\rangle$ and back.

[28]:
```
# This experiment uses these values from the previous experiment:
    # `qubit`,
    # `mem_slot`, and
    # `rough_qubit_frequency`.

# Rabi experiment parameters
num_rabi_points = 50

# Drive amplitude values to iterate over: 50 amplitudes evenly spaced from 0 to␣
↪0.75
drive_amp_min = 0
drive_amp_max = 0.75
drive_amps = np.linspace(drive_amp_min, drive_amp_max, num_rabi_points)
```

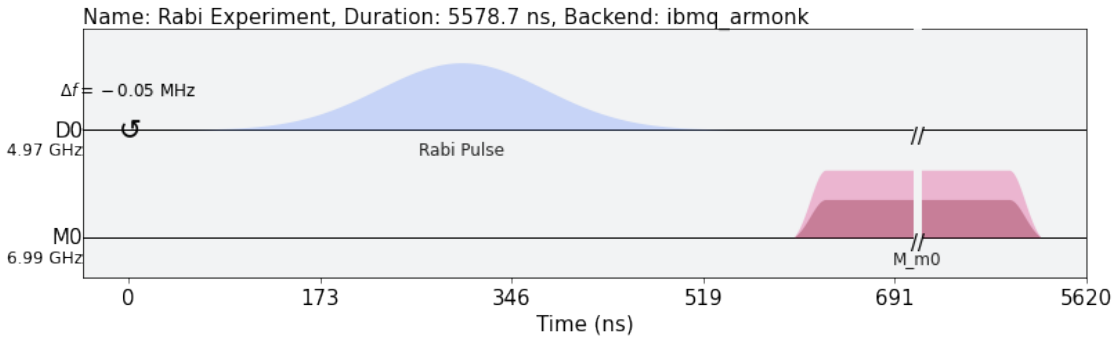
[29]:
```
# Build the Rabi experiments:
#    A drive pulse at the qubit frequency, followed by a measurement,
#    where we vary the drive amplitude each time.

drive_amp = Parameter('drive_amp')
with pulse.build(backend=backend, default_alignment='sequential', name='Rabi␣
↪Experiment') as rabi_sched:
    drive_duration = get_closest_multiple_of_16(pulse.
↪seconds_to_samples(drive_duration_sec))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.set_frequency(rough_qubit_frequency, drive_chan)
    pulse.play(pulse.Gaussian(duration=drive_duration,
```

```
                                amp=drive_amp,
                                sigma=drive_sigma,
                                name='Rabi Pulse'), drive_chan)
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])

rabi_schedules = [rabi_sched.assign_parameters({drive_amp: a}, inplace=False)␣
 ↪for a in drive_amps]
```

The schedule will look essentially the same as the frequency sweep experiment. The only difference is that we are running a set of experiments which vary the amplitude of the drive pulse, rather than its modulation frequency.

[30]:
```
rabi_schedules[-1].draw(backend=backend)
```

[30]:



[31]:
```
num_shots_per_point = 1024

job = backend.run(rabi_schedules,
                  meas_level=1,
                  meas_return='avg',
                  shots=num_shots_per_point)

job_monitor(job)
```

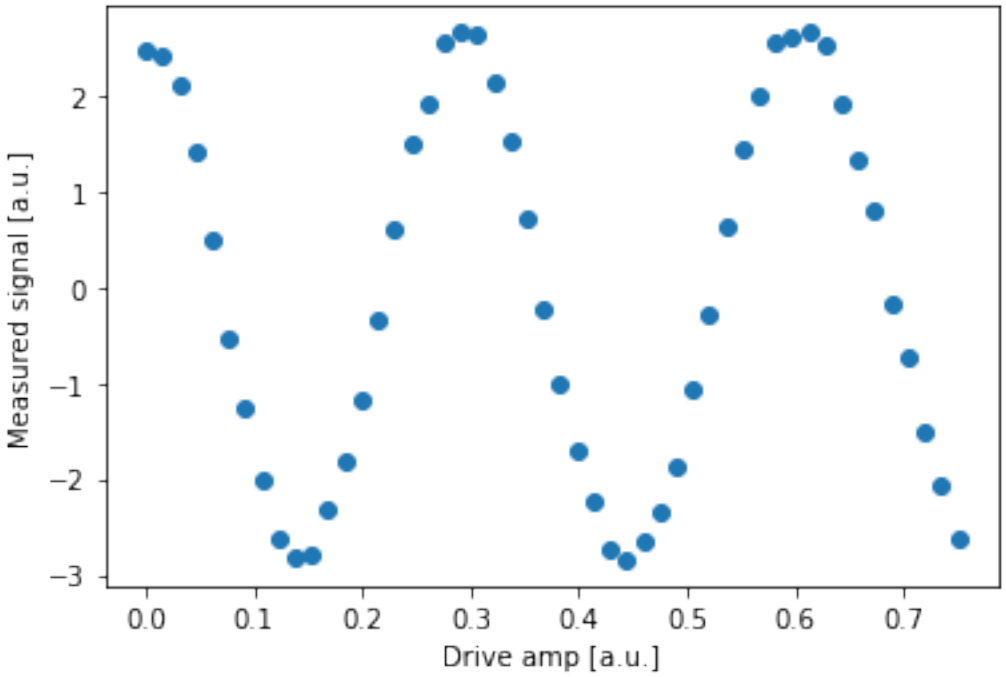
Job Status: job has successfully run

[32]:
```
rabi_results = job.result(timeout=120)
```

Now that we have our results, we will extract them and fit them to a sinusoidal curve.

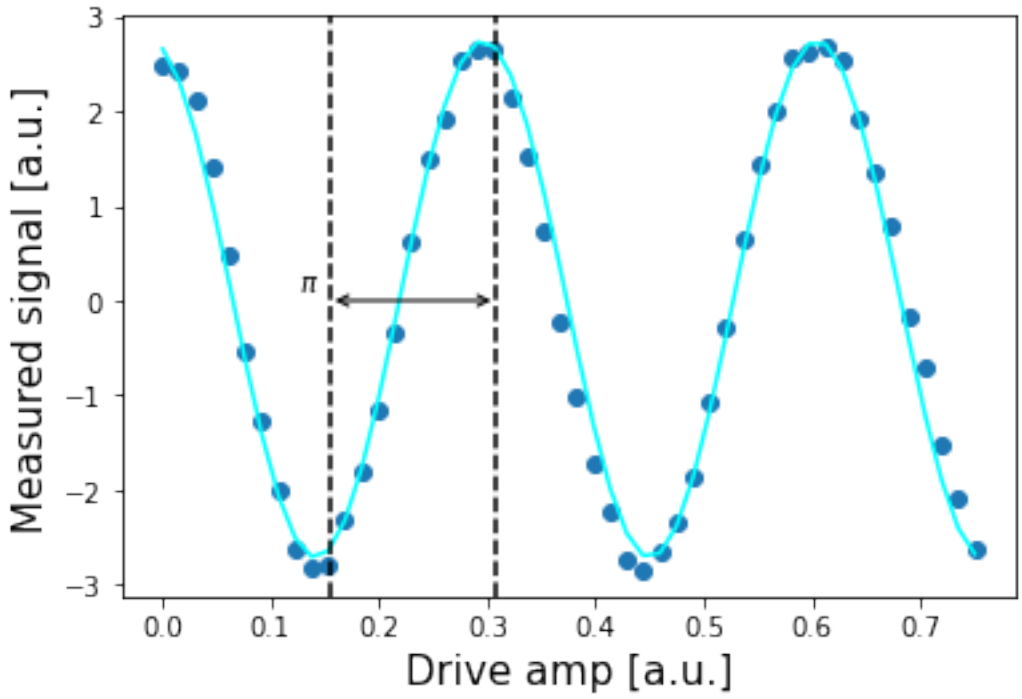
[33]:
```
# center data around 0
def baseline_remove(values):
    return np.array(values) - np.mean(values)
```

[35]:
```
rabi_values = []
for i in range(num_rabi_points):
    # Get the results for `qubit` from the ith experiment
    rabi_values.append(rabi_results.get_memory(i)[qubit] * scale_factor)

rabi_values = np.real(baseline_remove(rabi_values))

plt.xlabel("Drive amp [a.u.]")
plt.ylabel("Measured signal [a.u.]")
plt.scatter(drive_amps, rabi_values) # plot real part of Rabi values
plt.show()
```



[44]:
```
fit_params, y_fit = fit_function(drive_amps,
                                 rabi_values,
                                 lambda x, A, B, drive_period, phi: (A*np.
↪cos(2*np.pi*x/drive_period - phi) + B),
                                 [3, 0.1, 0.3, 0])

plt.scatter(drive_amps, rabi_values)
plt.plot(drive_amps, y_fit, color='cyan')

drive_period = fit_params[2] # get period of rabi oscillation

plt.axvline(drive_period/2, color='black', linestyle='--')
```

```
plt.axvline(drive_period, color='black', linestyle='--')
plt.annotate("", xy=(drive_period, 0), xytext=(drive_period/2,0),␣
 ↪arrowprops=dict(arrowstyle="<->", color='black'))
plt.annotate("$\pi$", xy=(drive_period/2-0.03, 0.1), color='black')

plt.xlabel("Drive amp [a.u.]", fontsize=15)
plt.ylabel("Measured signal [a.u.]", fontsize=15)
plt.show()
```



[45]:
```
pi_amp = abs(drive_period / 2)
print(f"Pi Amplitude = {pi_amp}")
```

```
Pi Amplitude = 0.1541941261155189
```

Let's define our pulse, with the amplitude we just found.

[46]:
```
with pulse.build(backend) as pi_pulse:
    drive_duration = get_closest_multiple_of_16(pulse.
 ↪seconds_to_samples(drive_duration_sec))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.play(pulse.Gaussian(duration=drive_duration,
                              amp=pi_amp,
                              sigma=drive_sigma,
```

```
                                name='pi_pulse'), drive_chan)
```

Appendix: Codes

Question 1

Code for Resonator 1

```
import matplotlib.pyplot as plt
import cmath
import numpy as np
import pandas as pd
from pandas import *
from scipy.optimize import curve_fit
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('Data1.csv', comment = '#', sep = ';', delimiter = ',')
frequency = data['X'].to_numpy()
freq = frequency/1000000
s21 = data['Y'].to_numpy()
for i in range(0,5000):
    s21[i] = complex(s21[i])
mag = np.absolute(s21)

#Test function with coefficients as parameters
def lorentzian(x, a, amp1, cen1, wid1):
    return (a+amp1*wid1**2/((x-cen1)**2+wid1**2))
par, cov = curve_fit(lorentzian,freq,mag,p0 = np.array((0.001,-1,5540,20)))
fit = lorentzian(freq,par[0],par[1],par[2],par[3])
print(par)

#Chi-Square
r = (mag-fit)
chisq = np.sum(r**2./fit)
print(chisq)

#3db S21 line
s3db = (max(fit)-min(fit))/np.sqrt(2)
#Plotting the data
plt.figure(1)
plt.plot(freq,mag,freq,fit)
plt.plot(freq,(min(fit)+s3db)*np.ones(len(freq)),'--')
plt.title(' Magnitude Plot for $S_{21}$ and Curve-fitting')
plt.ylabel('Magnitude')
plt.xlabel('Frequency (MHz)')
plt.legend(['Observed curve','Fitted curve','3 dB line'],loc = 'lower right')
plt.grid(True)
plt.show()
```

Code for Resonators 2, 3 and 4

```
import matplotlib.pyplot as plt
import cmath
import numpy as np
import pandas as pd
from pandas import *
from scipy.optimize import curve_fit
plt.rc('figure', figsize=(8, 4))

#Reading the .csv file
data = pd.read_csv('Data2.csv', comment = '#', sep = ';', delimiter = ',')
frequency = data['X'].to_numpy()
freq = frequency/1000000
s21_1 = data['0.00'].to_numpy()
for i in range(0,5000):
    s21_1[i] = complex(s21_1[i])
mag1 = np.absolute(s21_1)

s21_2 = data['1.00'].to_numpy()
for i in range(0,5000):
    s21_2[i] = complex(s21_2[i])
mag2 = np.absolute(s21_2)

s21_3 = data['2.00'].to_numpy()
for i in range(0,5000):
    s21_3[i] = complex(s21_3[i])
mag3 = np.absolute(s21_3)

s21_4 = data['3.00'].to_numpy()
for i in range(0,5000):
    s21_4[i] = complex(s21_4[i])
mag4 = np.absolute(s21_4)

s21_5 = data['4.00'].to_numpy()
for i in range(0,5000):
    s21_5[i] = complex(s21_5[i])
mag5 = np.absolute(s21_5)

s21_6 = data['5.00'].to_numpy()
for i in range(0,5000):
    s21_6[i] = complex(s21_6[i])
mag6 = np.absolute(s21_6)
```

```
s21_7 = data['6.00'].to_numpy()
for i in range(0,5000):
    s21_7[i] = complex(s21_7[i])
mag7 = np.absolute(s21_7)

#Test function with coefficients as parameters
def lorentzian(x, a, amp1, cen1, wid1):
    return (a+amp1*wid1**2/((x-cen1)**2+wid1**2))
par, cov = curve_fit(lorentzian,freq,mag1,p0 = np.array((0.03,-1,5894,1)))
fit = lorentzian(freq,par[0],par[1],par[2],par[3])

#Chi-Square
r = (mag1-fit)
chisq = np.sum(r**2./fit)
print(chisq)
print(par)
#Plotting the data
plt.figure(1)
plt.plot(freq,mag1,freq,mag2, freq, mag3, freq, mag4, freq, mag5, freq, mag6,
plt.title(' Magnitude Plot for $S_{21}$ for different powers')
plt.ylabel('Magnitude')
plt.xlabel('Frequency (MHz)')
plt.grid(True)

#3db S21 line
s3db = (max(fit)-min(fit))/np.sqrt(2)

plt.figure(2)
plt.plot(freq,mag1,freq,fit)
plt.plot(freq,(min(fit)+s3db)*np.ones(len(freq)),'--')
plt.title(' Magnitude Plot for $S_{21}$ and Curve-fitting')
plt.ylabel('Magnitude')
plt.xlabel('Frequency (MHz)')
plt.legend(['Observed curve','Fitted curve','3 dB line'],loc = 'lower right')
plt.grid(True)

plt.show()
```

Question 2

1. Generating Bell States

Bell State $\left|\phi^{+}\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

```
from qiskit import QuantumRegister, ClassicalRegister,
QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[1])
```

Bell State $\left|\psi^{+}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.x(qreg_q[1])
circuit.cx(qreg_q[0], qreg_q[1])
```

Bell State $\left|\phi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[1])
```

Bell State $\left|\psi^{-}\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
circuit.x(qreg_q[1])
circuit.cx(qreg_q[0], qreg_q[1])
```

2. Gate identities

   (a) HXH = Z

   HXH|0⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
```

   HXH|1⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
```

   Z|0⟩

```
 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.z(qreg_q[0])
```

   Z|1⟩

```
 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)
```

```
circuit.x(qreg_q[0])
circuit.z(qreg_q[0])
```

(b) HYH = -Y

HYH|0⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.y(qreg_q[0])
circuit.h(qreg_q[0])
```

HYH|1⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
circuit.y(qreg_q[0])
circuit.h(qreg_q[0])
```

Y|0⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.y(qreg_q[0])
```

Y|1⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)
```

```
circuit.x(qreg_q[0])
circuit.y(qreg_q[0])
```

(c) HZH = X

HXH|0⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.z(qreg_q[0])
circuit.h(qreg_q[0])
```

HXH|1⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
circuit.z(qreg_q[0])
circuit.h(qreg_q[0])
```

X|0⟩

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
```

X|1⟩

```
 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
```

```
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.x(qreg_q[0])
```

(d) HTH = $R_x(\frac{\pi}{4})$

HTH$|\mathbf{0}\rangle$

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.z(qreg_q[0])
circuit.h(qreg_q[0])
```

HTH$|\mathbf{1}\rangle$

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(1, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.x(qreg_q[0])
circuit.h(qreg_q[0])
circuit.t(qreg_q[0])
circuit.h(qreg_q[0])
```

$R_x(\frac{\pi}{4}|0\rangle$

$R_x(\frac{\pi}{4})|\mathbf{1}\rangle$

INDIAN INSTITUTE OF SCIENCE

IISC QUANTUM TECHNOLOGY INITIATIVE

QT211 - BASIC QUANTUM TECHNOLOGY LABORATORY

Assignment 4

Chaitali Shah
(SR No: 01-02-04-10-51-21-1-19786)

Date: November 29, 2023

Aim of the Experiment

1. Driven Oscillator: Drive an oscillator and include damping.

   (a) To find the steady state of the oscillator.

   (b) To determine the difference when drive is with Rotating Wave Approximation (RWA) or without RWA.

   (c) To Vary the strength of the drive and plot to find the steady-state.

2. Driven Two-Level system: Drive a TLS and include damping

   (a) To find the steady-state of the TLS.
   To determine the difference when drive is with Rotating Wave Approximation (RWA) or without RWA.

3. Anharmonic driven oscillator: $H_{(0)} = \Omega_t(\alpha \times num(N)^2 + num(N))$ where N is the number of levels you consider in the simulation.

   (a) To find the steady-state of the oscillator.

   (b) To determine the difference when drive is with Rotating Wave Approximation (RWA) or without RWA.

   (c) To Vary the strength of the drive and plot to find the steady-state.

1 Oscillator

Undriven Damped Oscillator

```
from qutip import *
from pylab import *
from scipy import *

# Define paramters
N = 20  # number of basis states to consider
a = destroy(N)
H = a.dag() * a
psi0 = basis(N, 10)  # initial state
kappa = 0.1  # coupling to oscillator

# collapse operators
c_op_list = []
n_th_a = 2  # temperature with average of 2 excitations
rate = kappa * (1 + n_th_a)
if rate > 0.0:
    c_op_list.append(sqrt(rate) * a)  # decay operators
rate = kappa * n_th_a
if rate > 0.0:
    c_op_list.append(sqrt(rate) * a.dag())  # excitation operators

# find steady-state solution
final_state = steadystate(H, c_op_list)
# find expectation value for particle number in steady state
fexpt = expect(a.dag() * a, final_state)

tlist = linspace(0, 50, 100)

# master eq.
medata = mesolve(H, psi0, tlist, c_op_list, [a.dag() * a])

plot(tlist, medata.expect[0], lw=2)
# plot steady-state expt. value as horizontal line (should be = 2)
axhline(y=fexpt, color='r', lw=1.5)
ylim([0, 10])
xlabel('Time', fontsize=14)
ylabel('Number of excitations', fontsize=14)
legend(('Master Equation', 'Steady State'))
title('Damped Harmonic Oscillator')
show()
```

A simple example of a system that reaches a steady state is a harmonic oscillator coupled to a thermal environment. Below we consider a harmonic oscillator, initially in the $|10\rangle$ number state, and weakly coupled to a thermal environment characterized by an average particle expectation value of $\langle n\rangle = 2$.

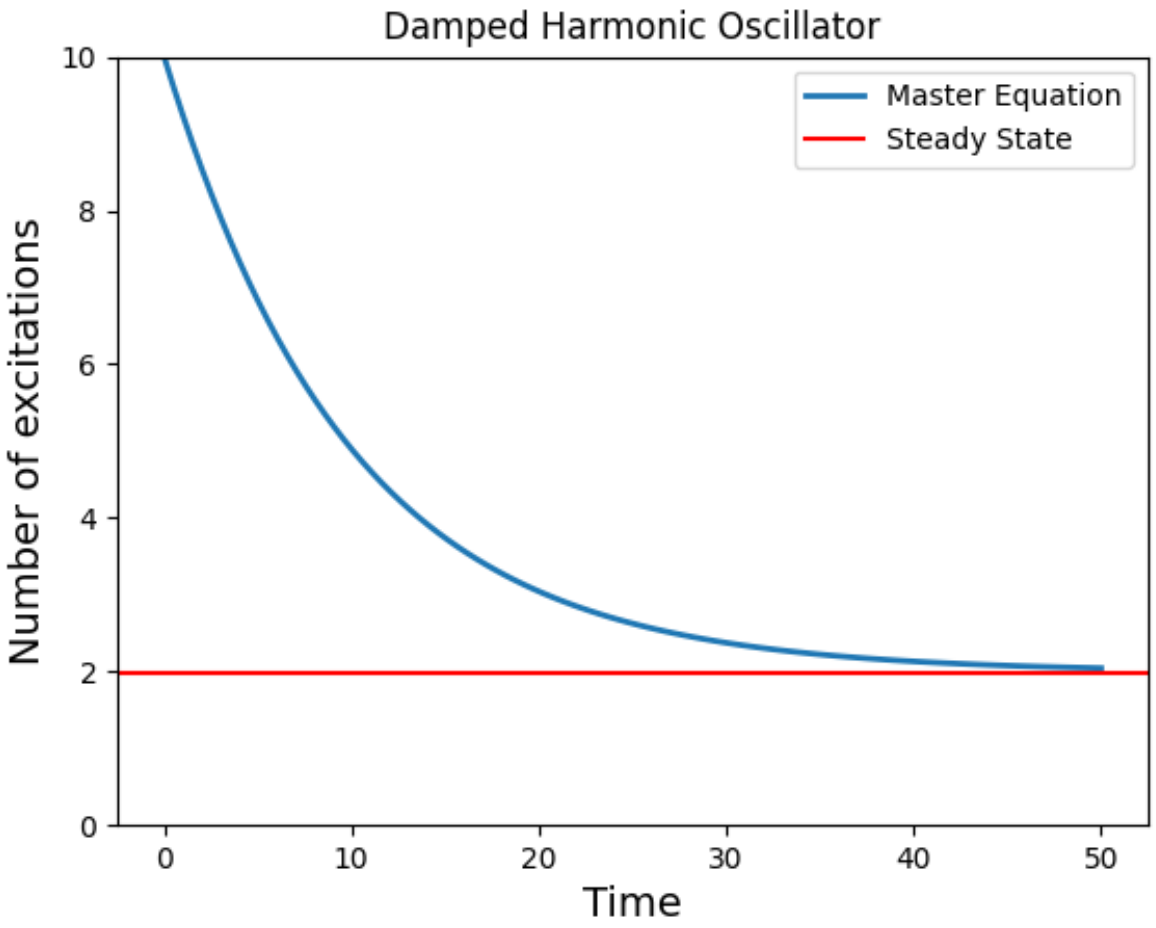


Figure 1: Damped Undriven Oscillator

Driven Damped Oscillator

```
from qutip import *
from pylab import *
from scipy import *

H = 2*np.pi * 0.1 * sigmax()
psi0 = basis(2, 0)
times = np.linspace(0.0, 10.0, 20)
result = sesolve(H, psi0, times, [sigmaz()])

result = sesolve(H, psi0, times, [sigmaz(), sigmay()])
result.expect

H = 2*np.pi * 0.1 * sigmax()
psi0 = basis(2, 0)
times = np.linspace(0.0, 10.0, 100)
result = sesolve(H, psi0, times, [sigmaz(), sigmay()])
fig, ax = plt.subplots()
ax.plot(result.times, result.expect[0])
ax.plot(result.times, result.expect[1])
ax.set_xlabel('Time')
ax.set_ylabel('Expectation values')
ax.legend(("Sigma-Z", "Sigma-Y"))
plt.show()

times = [0.0, 1.0]
result = mesolve(H, psi0, times, [], [])
result.states
```

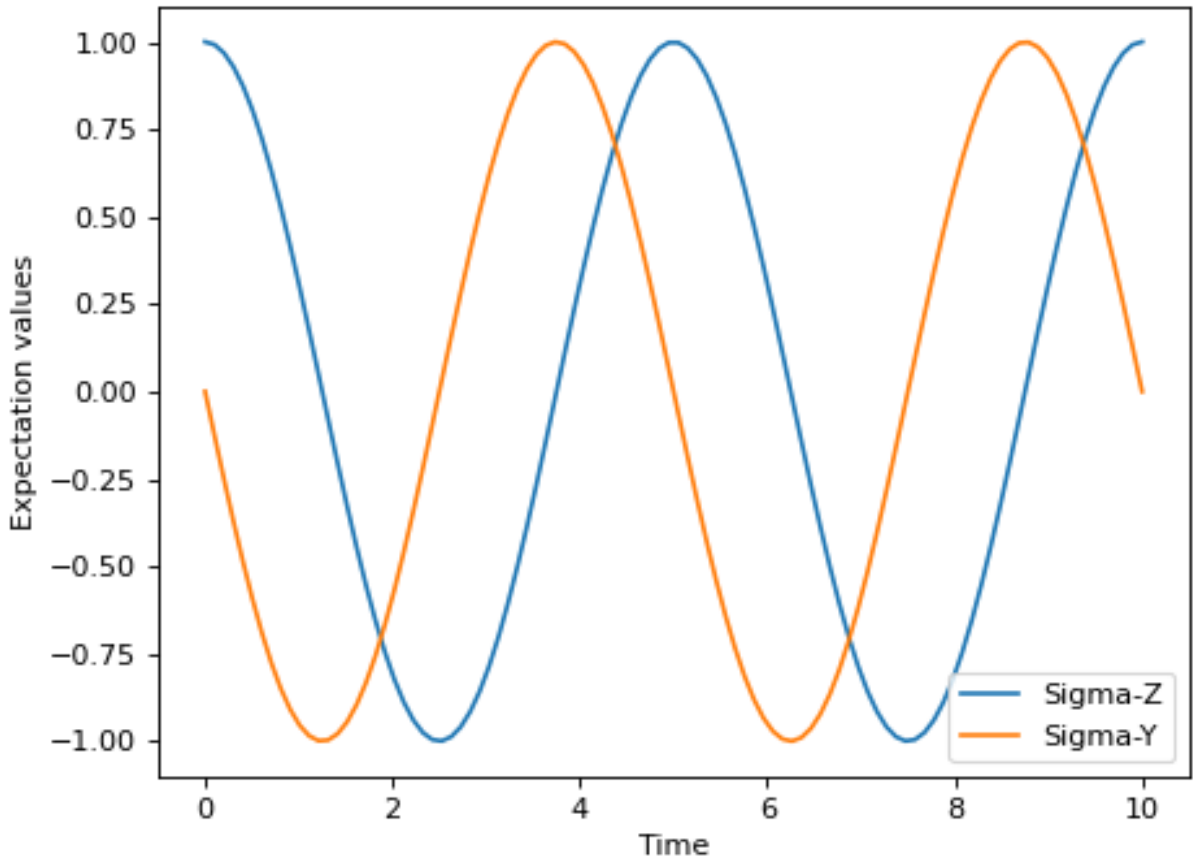


Figure 2: Driven Undamped Oscillator

Driven Damped Oscillator
We can easily add a relaxation process describing the dissipation of energy from the spin to its environment.

```
times = np.linspace(0.0, 10.0, 100)
result = mesolve(H, psi0,times,[np.sqrt(0.05)*sigmax()], [sigmaz(),sigmay()])
fig, ax = plt.subplots()
ax.plot(times, result.expect[0])
ax.plot(times, result.expect[1])
ax.set_xlabel('Time')
ax.set_ylabel('Expectation values')
ax.legend(("Sigma-Z", "Sigma-Y"))
plt.show()
```

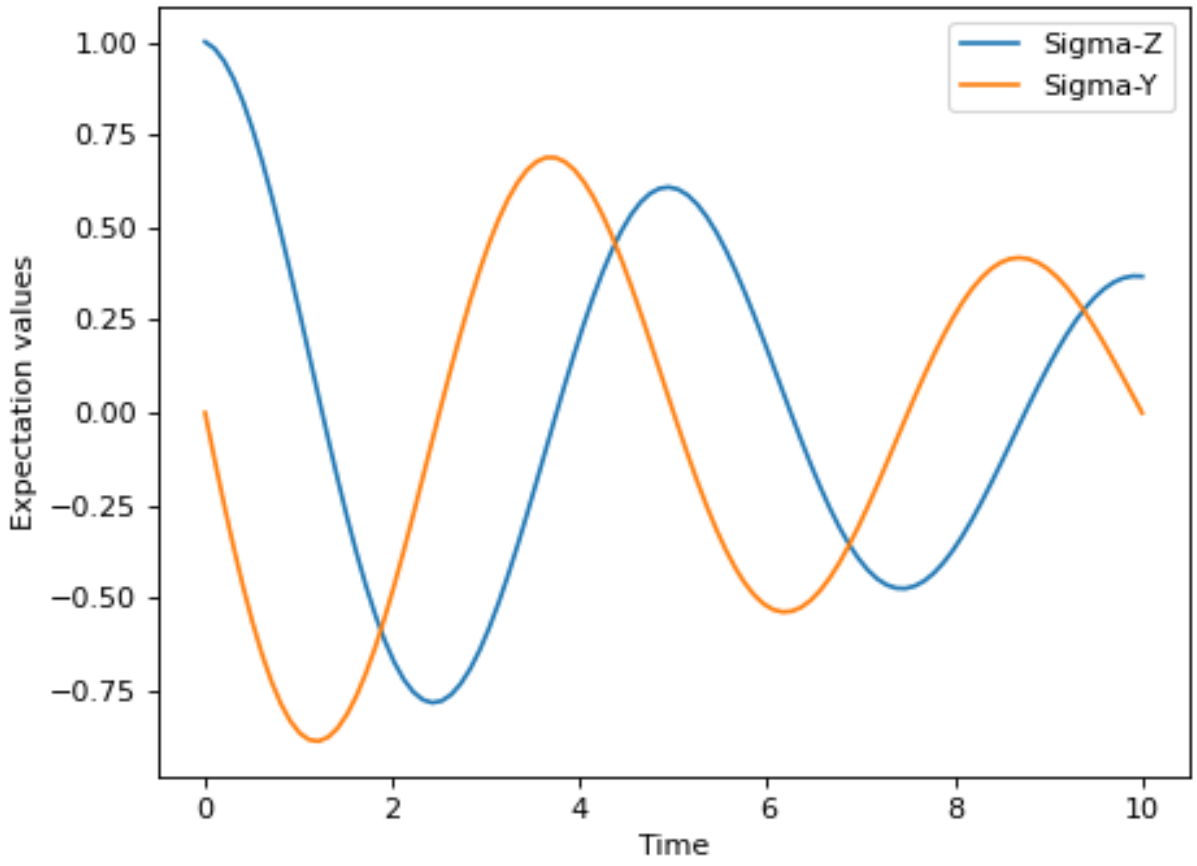


Figure 3: Driven and Damped Oscillator

Rotating Wave Approximation For weak coupling and low temperature the Rotating Wave Approximation (RWA) performs better, whereas for finite coupling and higher bath-temperature the truncated master equation is favourable. RWA fails, in particular for non-equilibrium scenarios like transient dynamics and non-equilibrium steady states which are non-trivial also in the high-temperature regime.

2 Two Level System

Consider a two-level atom coupled to a leaky single-mode cavity through a dipole-type interaction, which supports a coherent exchange of quanta between the two systems. If the atom initially is in its groundstate and the cavity in a 5-photon Fock state, the dynamics is calculated with the lines following code:

Code for a Two Level System

```
times = np.linspace(0.0, 10.0, 200)
psi0 = tensor(fock(2,0), fock(10, 5))
a  = tensor(qeye(2), destroy(10))
sm = tensor(destroy(2), qeye(10))
H = 2*np.pi*a.dag()*a +2*np.pi*sm.dag()*sm +2*np.pi*0.25*(sm*a.dag()+ sm.dag()
result = mesolve(H, psi0, times, [np.sqrt(0.1)*a], [a.dag()*a, sm.dag()*sm])
plt.figure()
plt.plot(times, result.expect[0])
plt.plot(times, result.expect[1])
plt.xlabel('Time')
plt.ylabel('Expectation values')
plt.legend(("cavity photon number", "atom excitation probability"))
plt.show()
```

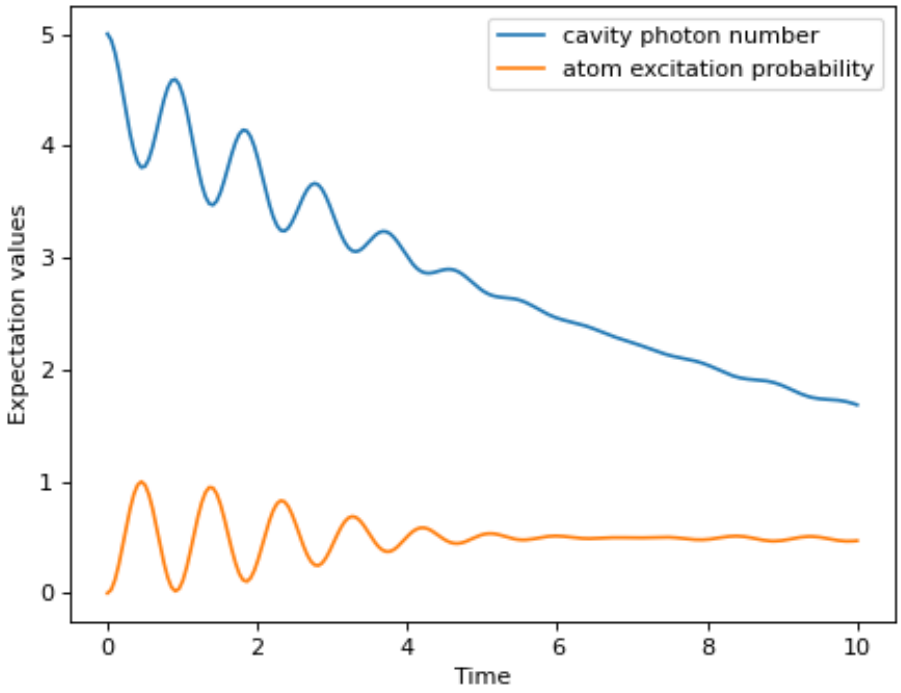


Figure 4: Two Level System Steady State

Vaccuum Rabi Oscillations We use the following code

```
n_th_a = 0.0          # avg number of thermal bath excitation
use_rwa = True      # True for RWA and False for Without RWA

tlist = np.linspace(0,25,101)

# intial state
psi0 = tensor(basis(N,0), basis(2,1))    # start with an excited atom

# operators
a  = tensor(destroy(N), qeye(2))
sm = tensor(qeye(N), destroy(2))

# Hamiltonian
if use_rwa:
    H = wc*a.dag()*a + wa*sm.dag()*sm + g*(a.dag()*sm + a *sm.dag())
else:
    H = wc*a.dag()*a + wa*sm.dag()*sm + g*(a.dag() + a)*(sm + sm.dag())

c_ops = []

# cavity relaxation
rate = kappa * (1 + n_th_a)
if rate > 0.0:
    c_ops.append(np.sqrt(rate) * a)

# cavity excitation, if temperature > 0
rate = kappa * n_th_a
if rate > 0.0:
    c_ops.append(np.sqrt(rate) * a.dag())

# qubit relaxation
rate = gamma
if rate > 0.0:
    c_ops.append(np.sqrt(rate) * sm)

output = mesolve(H, psi0, tlist, c_ops, [a.dag() * a, sm.dag() * sm])

n_c = output.expect[0]
n_a = output.expect[1]

fig, axes = plt.subplots(1, 1, figsize=(10,6))
```

```
axes.plot(tlist, n_c, label="Cavity")
axes.plot(tlist, n_a, label="Atom excited state")
axes.legend(loc=0)
axes.set_xlabel('Time')
axes.set_ylabel('Occupation probability')
axes.set_title('Vacuum Rabi oscillations')
plt.grid()
plt.show()
```

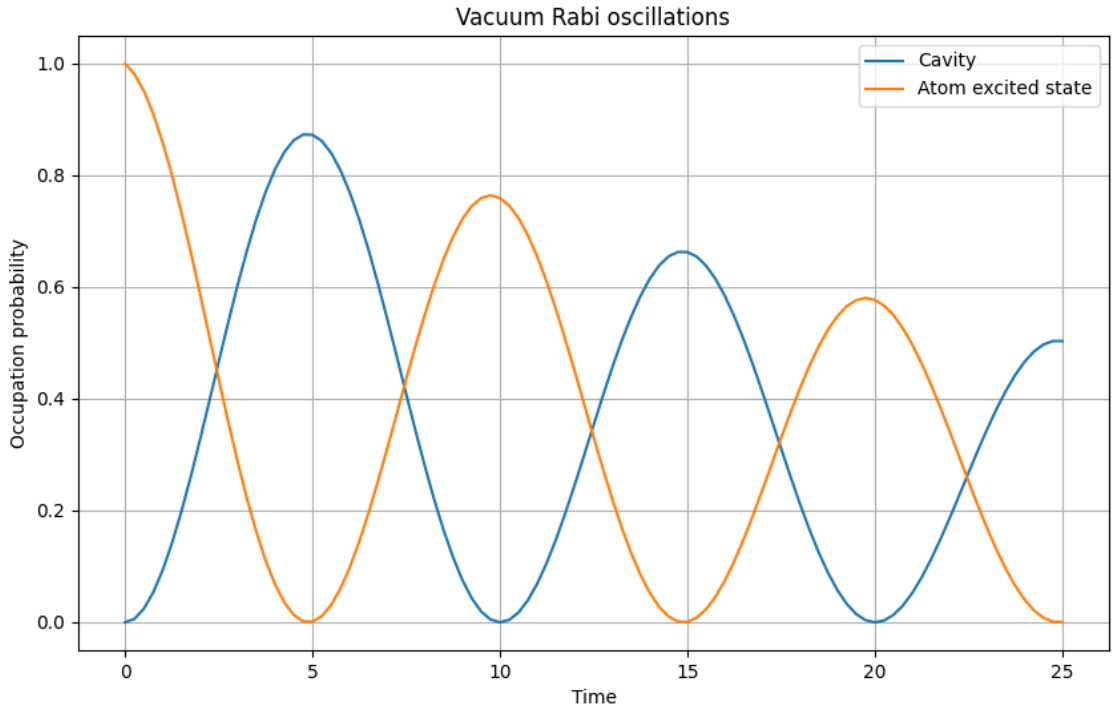


Figure 5: TLS with RWA

Rotating Wave Approximation A Two-Level System is a particularly simple quantum system. As such, some of the shortcomings of the RWA are obscured in this case. We find that the RWA gives the correct equilibrium state for a thermal environment in this case, in addition to the correct timescales, though it may miss some of the corrections to the transient quantum evolution that can be obtained from the weak-coupling master equation without the RWA.

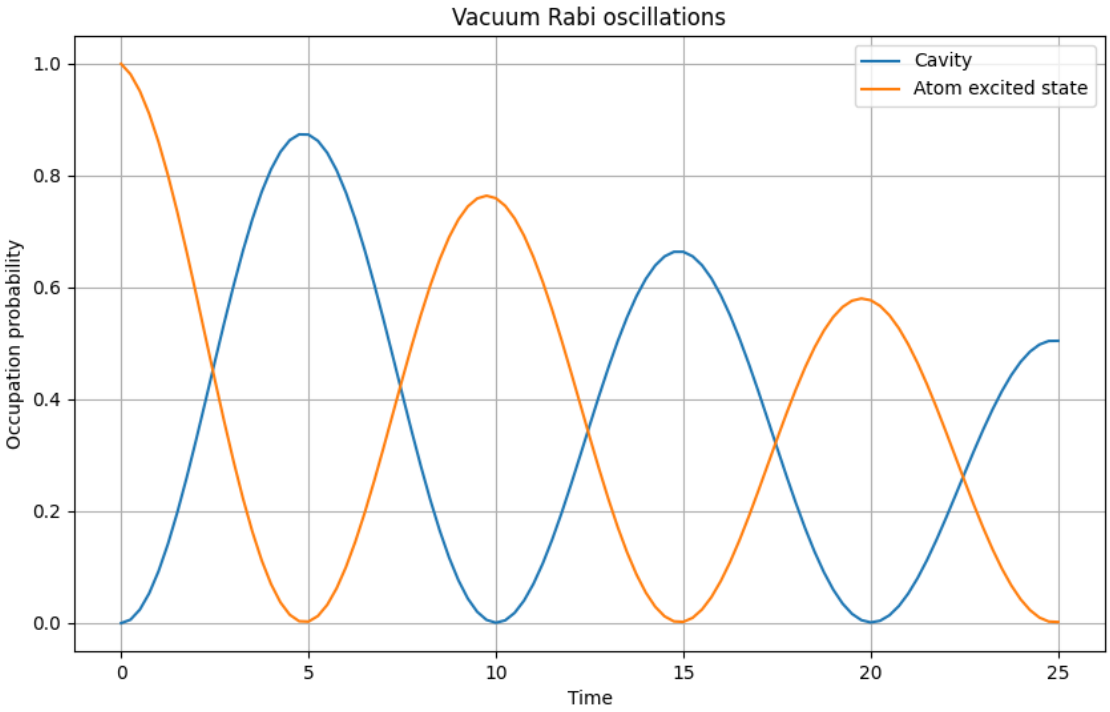


Figure 6: TLS without RWA

3 Anharmonic Oscillator

We can use a similar code to the one used in Assignment 3 to obtain the steady state for Anharmonic oscillator. The variation of output with a change in drive is shown below:

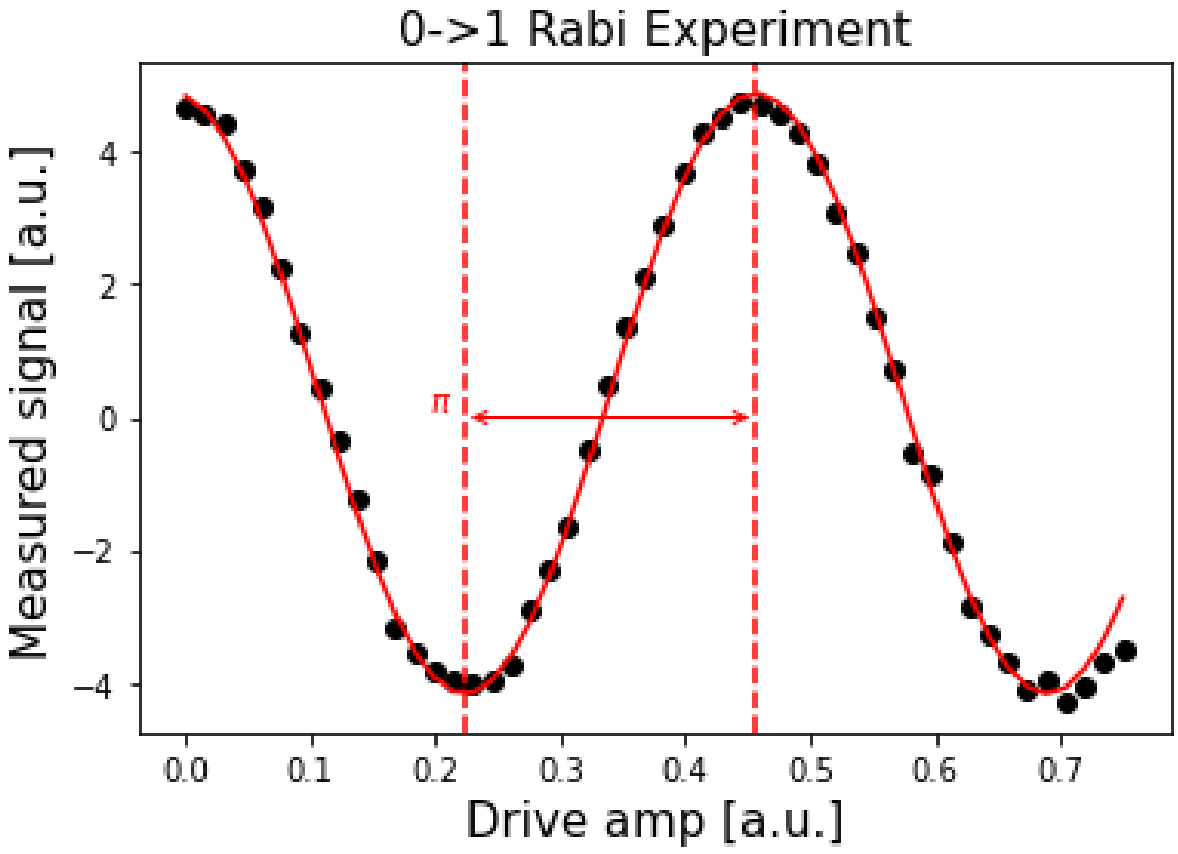


Figure 7: Rabi oscillations for varyin drive⟩

Explanation The Harmonic Oscillator has even-spaced energy levels and the transmon (anharmonic oscillator) does not, which is why we can use it as a qubit.

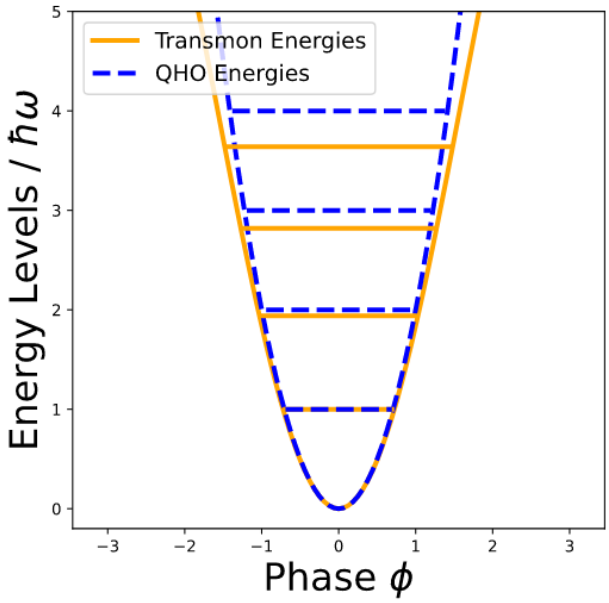


Figure 8: Energy level of a harmonic and anharmonic oscillator

In an anharmonic oscillator, the energy levels become less widely spaced at high excitation. In a Two Level system only two levels of an oscillator are considered.