The Laplace Transform

Assignment 6

EE2703 - Applied Programming Lab

Abhigyan Chattopadhyay
EE19B146

18th April 2021

Contents

1 The Problem at Hand **2**
1.1 The Laplace Transform 2

2 Solutions to given Questions **2**
2.1 Question 1: Impulse Response of a Damped Spring System 2
2.2 Question 2: Impulse Response with smaller decay 5
2.3 Question 3: Simulating the Impulse Response Calculation using `lsim` 6
2.4 Question 4: Coupled Spring Problem 7
2.5 Question 5: Bode Plots of 2 Port Network 9
2.6 Question 6: Transient Response of 2 Port Network with given Inputs 11

3 Conclusions **13**

1 The Problem at Hand

We want to analyse the frequency responses of various given input signals using the `scipy.signal` module in python.

1.1 The Laplace Transform

The Laplace Transform of a function takes the function in the time domain and converts it to its frequency domain representation in the Complex plane.

The equation for the (bilateral) Laplace Transform of a function $f(x)$ is:

$$F(s) = \int_{-\infty}^{\infty} f(t)e^{-st}dt$$

It also has some properties, of which we will be using the Derivative Property later on.

2 Solutions to given Questions

2.1 Question 1: Impulse Response of a Damped Spring System

We are given that a spring satisfying the equation:

$$\ddot{x} + 2.25x = f(t)$$

And it is acted upon by the forcing function:

$$f(t) = \cos(1.5t)e^{-0.5t}u_0(t)$$

The Laplace Transform of the given forcing function is given by:

$$F(s) = \frac{s + 0.5}{(s + 0.5^2) + 2.25}$$

We will try to solve this over the time interval from 0 to 50 seconds using the `system.impulse` function as follows:

```
def main():
    F1 = transfer(0.5,1.5)
    X1 = LTIMore([1,0,2.25],[1])
    H1 = F1/X1
    times = np.linspace(0,50,1000)
    ts,xs = sp.impulse(H1,None,times)
    oneshotplot(1,ts,xs,title="Plot for Question 1")
```

Note that some custom functions and classes were used here, which are defined in the `Assignment_6.py` file. They are also given here for reference and they are explained in the comments:

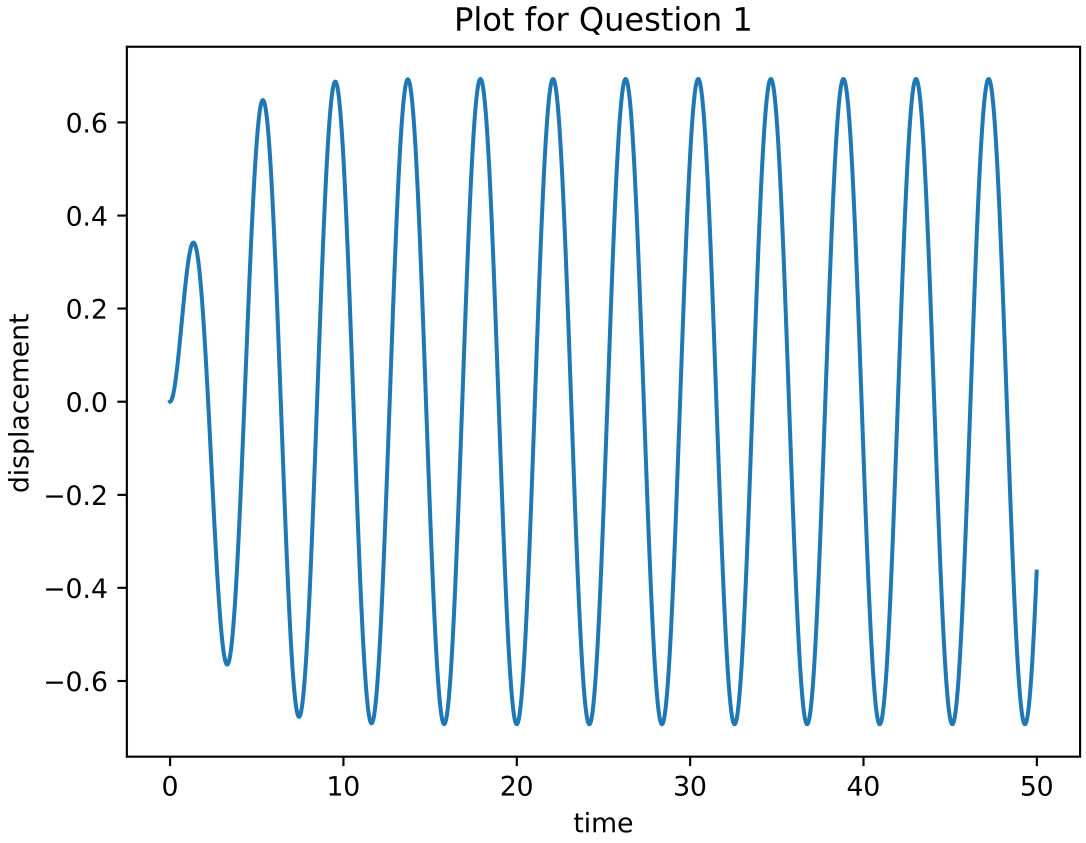
```
import numpy as np
import scipy.signal as sp
import matplotlib.pyplot as plt
from scipy.signal.ltisys import TransferFunctionContinuous as TransFun
class LTIMore(TransFun):
    def __neg__(self):
        return LTIMore(-self.num,self.den)
```

```
    def __mul__(self,other):
        if type(other) in [int,float]:
            return LTIMore(self.num*other,self.den)
        elif type(other) in [TransFun,LTIMore]:
            n = np.polymul(other.num,self.num)
            d = np.polymul(other.den,self.den)
            return LTIMore(n,d)

    def __truediv__(self,other):
        if type(other) in [int,float]:
            return LTIMore(self.num,self.den*other)
        elif type(other) in [TransFun,LTIMore]:
            n = np.polymul(other.den,self.num)
            d = np.polymul(other.num,self.den)
            return LTIMore(n,d)

    def __rtruediv__(self,other):
        if type(other) in [int,float]:
            return LTIMore(self.den*other,self.num)
        elif type(other) in [TransFun,LTIMore]:
            n = np.polymul(self.den,other.num)
            d = np.polymul(self.num,other.den)
            return LTIMore(n,d)

    def __add__(self,other):
        if type(other) in [int,float]:
            return LTIMore(np.polyadd(self.num,other*self.den),self.den)
        elif type(other) in [TransFun,LTIMore]:
            n = np.polyadd(np.polymul(self.den,other.num),np.polymul(self.num,
    ↪ other.den))
            d = np.polymul(self.den,other.den)
            return LTIMore(n,d)

    def __sub__(self,other):
        return self + (-other)
    def __rsub__(self,other):
        return (-self) + other

def transfer(alpha,omega):
    return LTIMore([1,alpha],[1,2*alpha,omega**2+alpha**2])

def oneshotplot(i,x,y,xlabel='t',ylabel='x',title="Fig"):
    """To Plot Graphs in one single function"""
    plt.figure(i)
    plt.plot(x,y,"-r")
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.savefig("fig"+str(i),dpi=1000)
    plt.show()
```

In essence, the LTIMore class adds the functionality of addition, subtraction, multiplication and division to the vanilla `scipy.signal.lti.TransferFunctionContinuous` class and allows for easy manipulation of the transfer functions.

The `oneshotplot` function allows us to plot a graph with a specified title and labels all in one line instead of calling each individual `pyplot` command.

We obtain the following graph:



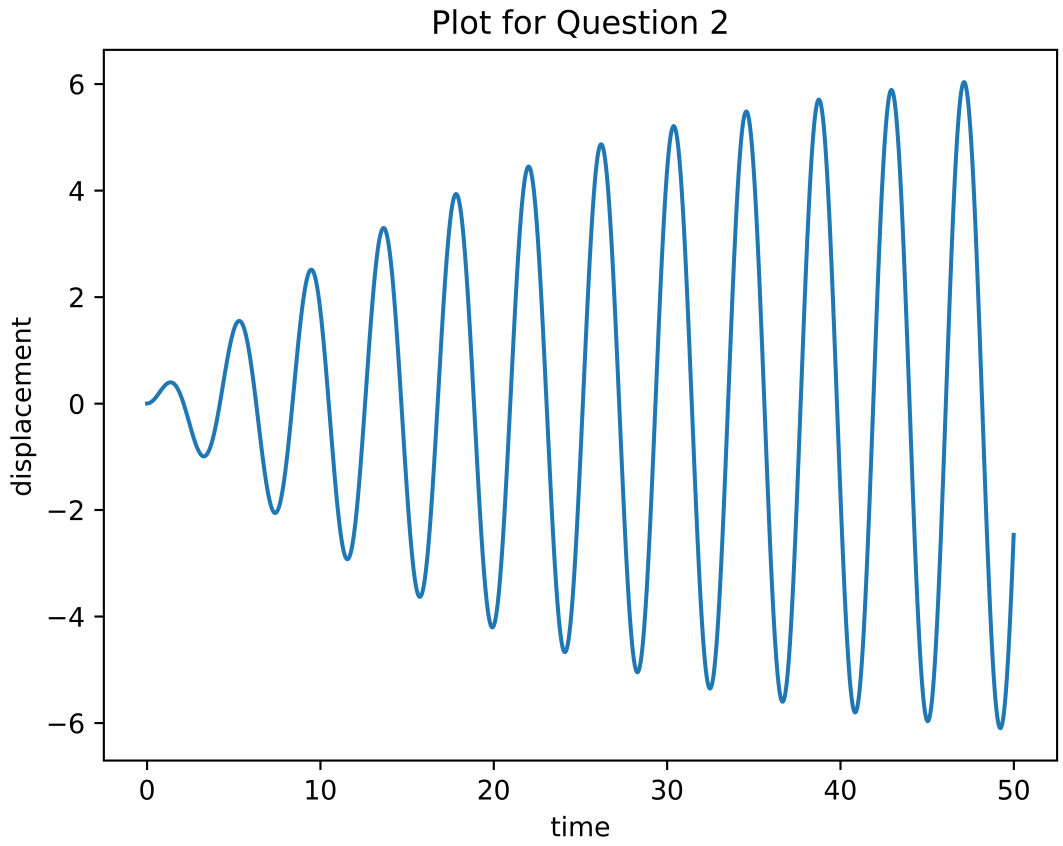
As we can see, it is a sinusoidal graph that initially starts with some damping but eventually it gets into a steady state of oscillations.

2.2 Question 2: Impulse Response with smaller decay

The code for this is similar, we just need to change a few of the values that we send into our functions.

```
def main():
    ...
    F2 = transfer(0.05,1.5)
    H2 = F2/X1
    ts2,xs2 = sp.impulse(H2,None,times)
    oneshotplot(2,ts2,xs2,'time','displacement',title="Plot for Question 2")
```

And we obtain the following graph:

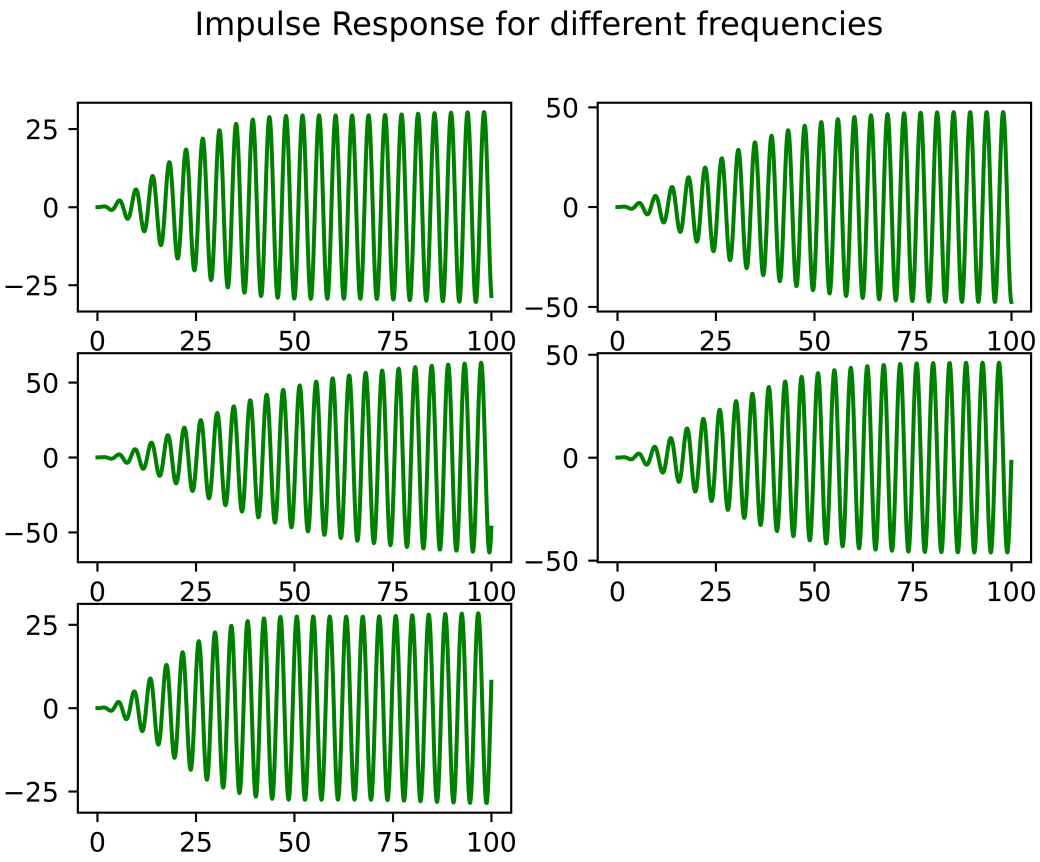


2.3 Question 3: Simulating the Impulse Response Calculation using `lsim`

Here we loop over the different mentioned frequencies and plot each of them as a subplot.

```
def main():
    ...
    myrange = np.arange(1.4,1.61,0.05)
    times1 = np.linspace(0,100,1000)
    plt.figure(3)
    for i,f in enumerate(myrange):
        currf = (np.cos(f*times1)*np.exp(-0.05*times1))*np.heaviside(times,0)
        _,y,_ = sp.lsim(H2,currf,times1)
        plt.subplot(3,2,i+1)
        plt.plot(times1,y,"-g")
    plt.suptitle("Impulse Response for different frequencies")
    plt.savefig("images/fig3.png",dpi=1000)
    plt.show()
```

And the plots obtained are as follows:



2.4 Question 4: Coupled Spring Problem

First we need to convert the given differential equation into its corresponding Laplace Transform.

The given equations are $\ddot{x} + (x - y) = 0$ and $\ddot{y} + 2(y - x) = 0$, with initial conditions: $x(0) = 1$, $\dot{x}(0) = y(0) = \dot{y}(0) = 0$

We take the Laplace Transform of these equations and we obtain:

$$s^2X(s) - s + X(s) - Y(s) = 0 - (i)$$

$$s^2Y(s) + 2Y(s) - 2X(s) = 0 - (ii)$$

On substituting $Y(s)$ from the first equation into the second, we obtain:

$$X(s) = \frac{s^2 + 2}{s^3 + 3s}$$

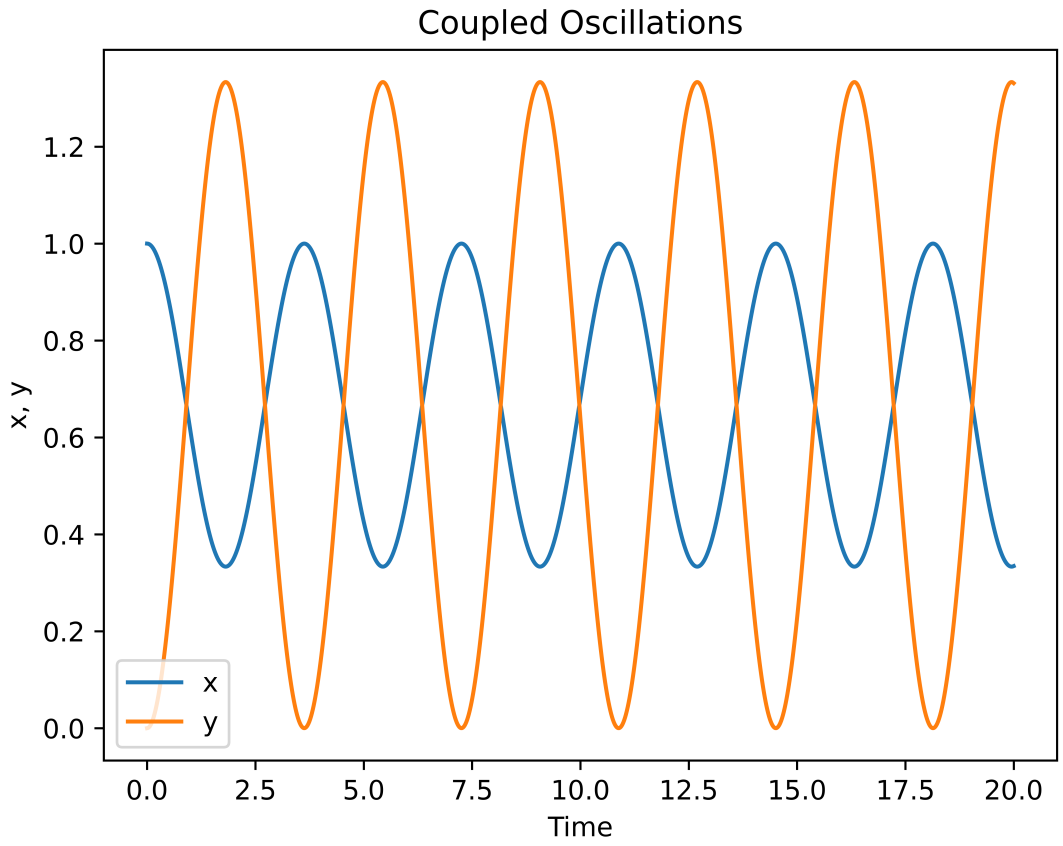
and

$$Y(s) = \frac{2}{s^3 + 3s}$$

Hence, we now solve the two individually using the `impulse` function and plot them both on the same graph:

```
def main():
    ...
    times2 = np.linspace(0,20,1000)
    Hx = LTIMore([1,0,2],[1,0,3,0])
    Hy = LTIMore([2],[1,0,3,0])
    tsx,xsx = sp.impulse(Hx,None,times2)
    tsy,xsy = sp.impulse(Hy,None,times2)
    plt.figure(4)
    plt.plot(tsx,xsx,label='x')
    plt.plot(tsy,xsy,label='y')
    plt.xlabel("Time")
    plt.ylabel("x, y")
    plt.legend()
    plt.title("Coupled Oscillations")
    plt.savefig("images/fig4.png",dpi=1000)
    plt.show()
```

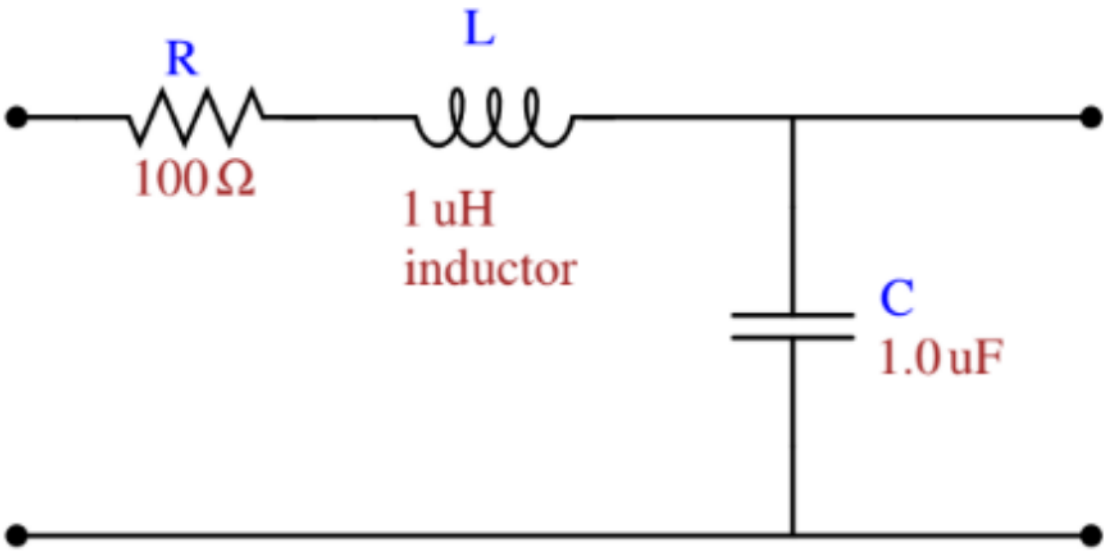
And we obtain the following graph:



We see that the two curves are of the same frequency and seem to be exactly 180 deg apart in phase and have different amplitudes.

2.5 Question 5: Bode Plots of 2 Port Network

We are given the following 2 port network:

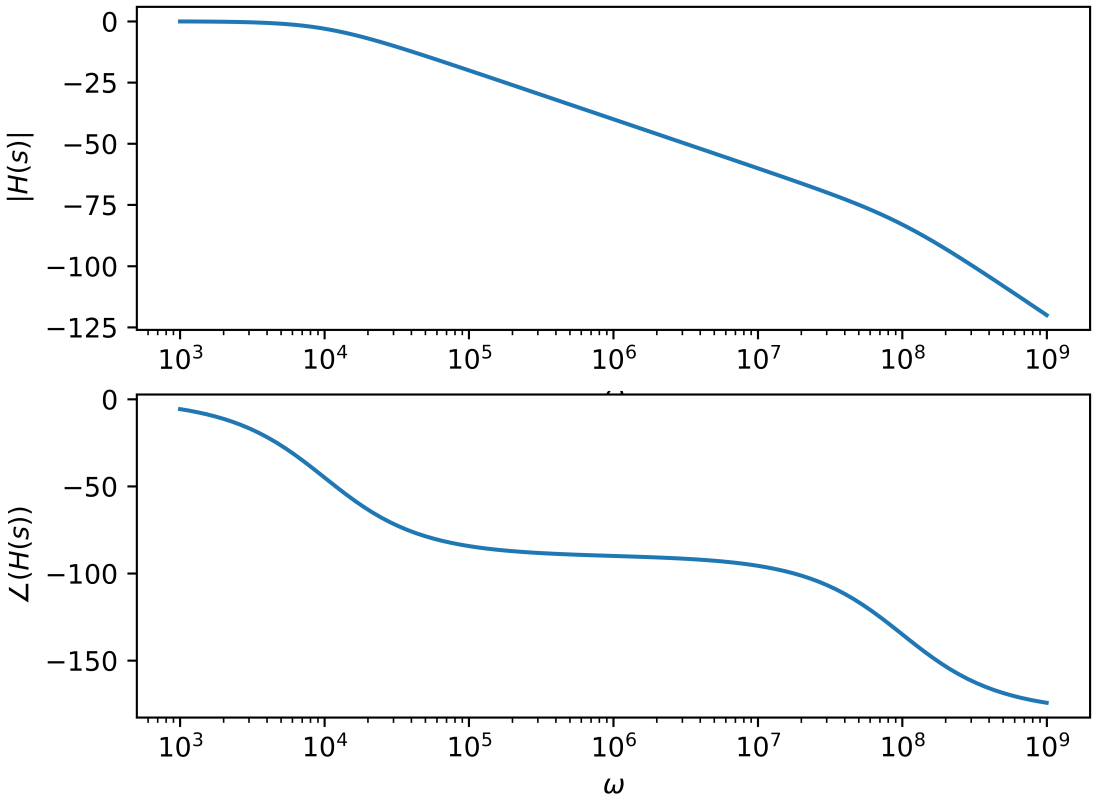


Once we solve the given 2 Port network, we obtain the following transfer function:

$$H(s) = \frac{10^6}{10^{-6}s^2 + 100s + 10^6}$$

And then we obtain the Bode Plot of this.

```
def main():
    ...
    H2p = LTIMore([1000000],[0.000001,100,1000000])
    w,mag,phase = sp.bode(H2p)
    plt.figure(5)
    plt.subplot(2,1,1)
    plt.semilogx(w,mag)
    plt.xlabel(r"$\omega$")
    plt.ylabel(r"$|H(s)|$")
    plt.subplot(2,1,2)
    plt.semilogx(w,phase)
    plt.xlabel(r"$\omega$")
    plt.ylabel(r"$\angle(H(s))$")
    plt.suptitle("Bode Plots for 2 Port Network")
    plt.savefig("images/fig5.png",dpi=1000)
    plt.show()
```


Bode Plots for 2 Port Network
0
−25
−50
−75
−100
−125
|H(s)|
10^3
10^4
10^5
10^6
10^7
10^8
10^9
0
−50
−100
−150
∠(H(s))
10^3
10^4
10^5
10^6
10^7
10^8
10^9
ω

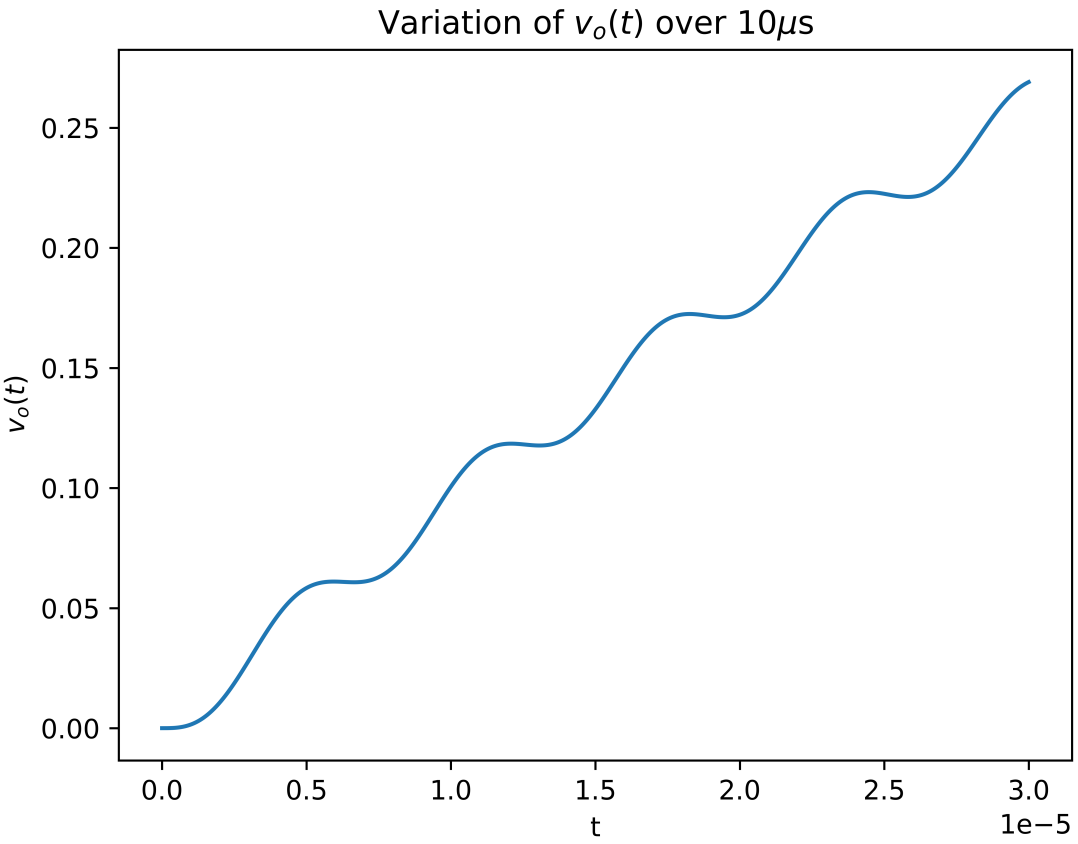
2.6 Question 6: Transient Response of 2 Port Network with given Inputs

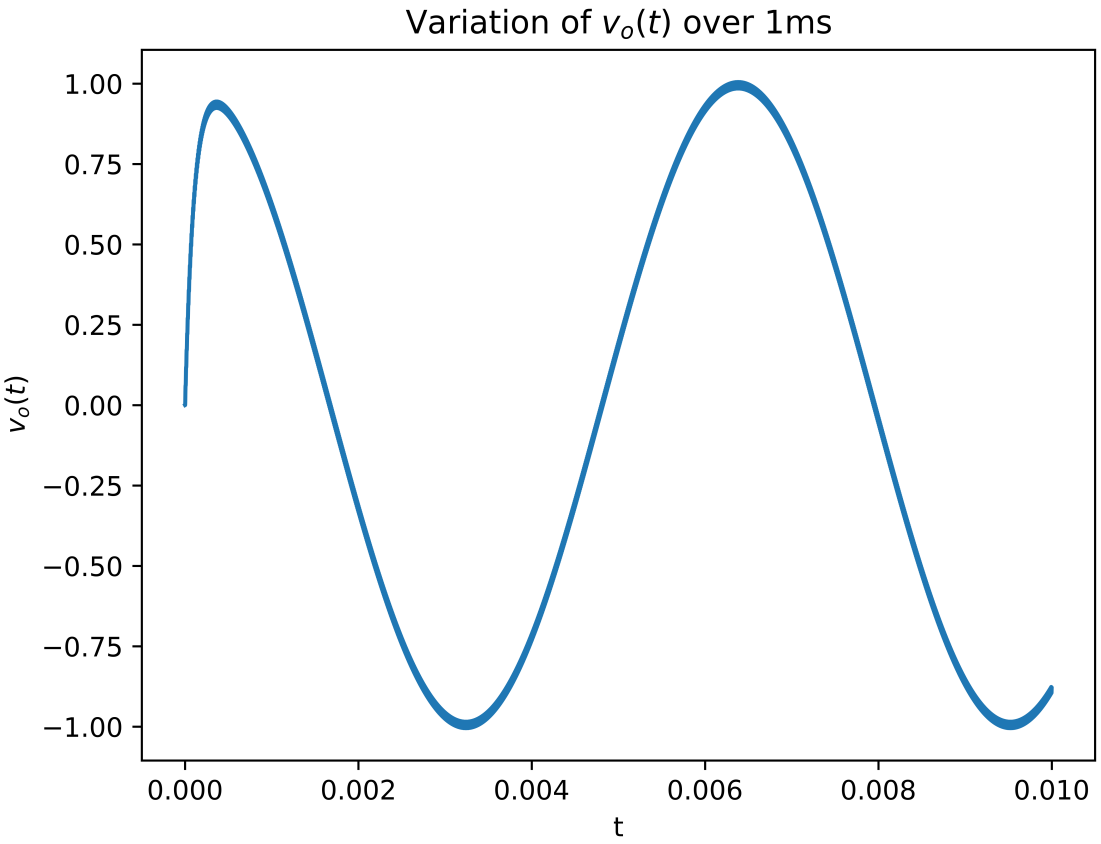
We are given the input function which we will use to simulate the response given the `lsim` function, and then see both the short term and long term responses.

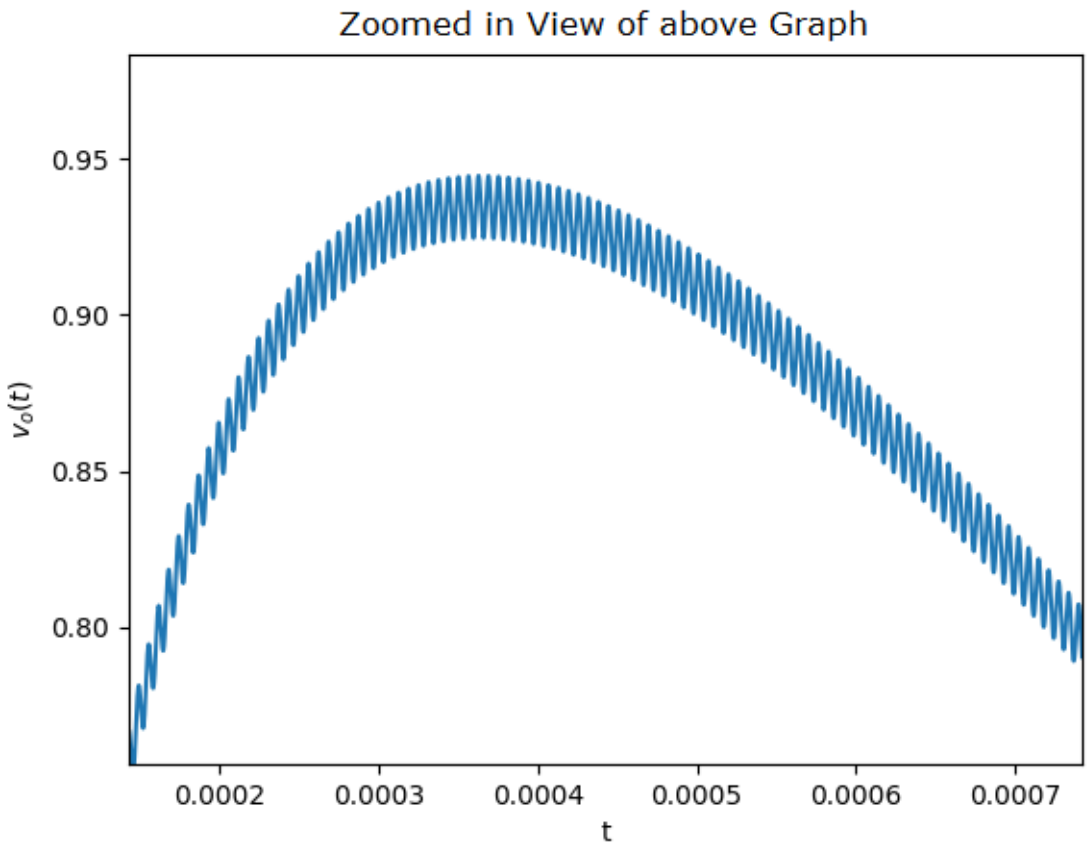
We will simulate the function over 10 μs for the short term response and over 1ms for the long term response.

```
def main():
    ...
    times3 = np.linspace(0,30*0.000001,1000)
    vi = np.multiply(np.cos(1000*times3)-np.cos(1000000*times3),np.heaviside(
    ↪ times3,0))
    _,y1,_ = sp.lsim(H2p,vi,times3)
    oneshotplot(6,times3,y1,'t',r'$v_{o}(t)$',r"Variation of $v_o(t)$ over 10$\
    ↪ mu$s")
    times4 = np.linspace(0,10*0.001,100000)
    vi = np.multiply(np.cos(1000*times4)-np.cos(1000000*times4),np.heaviside(
    ↪ times4,0))
    _,y2,_ = sp.lsim(H2p,vi,times4)
    oneshotplot(7,times4,y2,'t',r'$v_{o}(t)$',r"Variation of $v_o(t)$ over 1ms")
```

And we obtain the following graphs:




Variation of $v_o(t)$ over 1ms
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$v_o(t)$
0.000
0.002
0.004
0.006
0.008
0.010
t



The Figure 7 is made of tiny oscillating values which are visible when we zoom in. Their peak to peak values are about 0.02, and they are damped by the system. Effectively, the given RLC Circuit functions as a Low Pass filter and damps out the higher frequencies while keeping the low frequencies intact.

3 Conclusions

- `scipy` provides a lot of useful signal processing functionality through the `signal` sub-module.
- We used some of the available functions to analyze various systems, both mechanical and electrical
- We obtained Bode Plots and implemented a crude Low Pass filter with the given RLC Circuit that dampened the high frequencies.