EE2703: Applied Programming Lab
Assignment 6
The Laplace Transform

Kaushik Ravibaskar
EE20B057

March 16, 2022

Contents

1 Aim

The assignment mainly deals with the following points:

- To solve higher order systems using **Laplace Transforms**.
- To understand various functions such as **impulse** to take inverse and **lsim** to take convolution, present in **scipy.signal** module to solve Laplace based equations.
- To plot the results or simulations of mechanical and electrical problems in a meaningful manner and comment upon the nature of corresponding behaviour.

2 Introduction

This assignment mainly comprises of solving mechanical and electrical problems in **Laplace** domain. **Laplace Transform** is a very powerful mathematical tool to solve differential equations. It converts differential equations into simple algebraic equations which makes computation fast and easy. All differential equations signify **Linear Time Invariant** systems, hence it incorporates the use of Laplace transform.
In the following portions, we will solve problems using **Laplace transform** and use **Python functions** to compute and plot the results.

3 The Assignment Problems

3.1 P1: Solving a spring system

We are given the following spring system,

$$\frac{d^2x(t)}{dt^2} + 2.25x(t) = f(t)$$

where $f(t)$ is given by,

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

Its Laplace transform can be given as,

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

Now, when we convert differential equations into Laplace equations,

$$\frac{dx(t)}{dt} \longrightarrow sX(s) - x(0^-)$$

using this conversion, and the fact that $x(0) = 0$ and $x'(0) = 0$, we get the following Laplace equation,

$$s^2X(s) + 2.25X(s) = F(s)$$

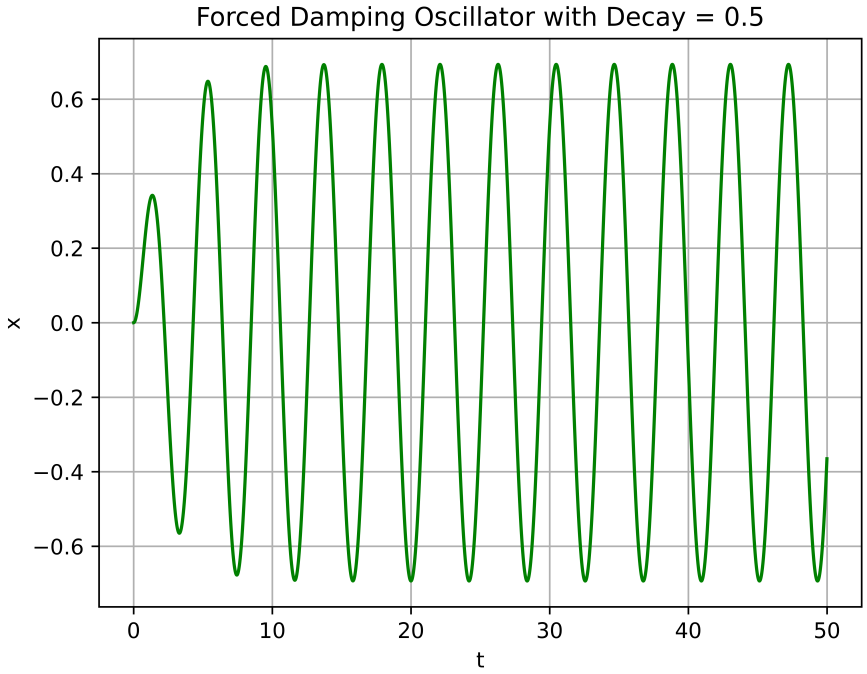
thus we can say that,

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

We can use the python function **lti** to construct the rational function $X(s)$ and another function **impulse** to compute the **time domain** response of $X(s)$. The code to do this process and plot the results is as follows:

```
#defining a function to compute laplace function of x(t)
def trans_x(freq, decay):
    p_den_x = np.polymul([1.0, 0, 2.25], [1, 2*decay
    , (decay)*(decay) + (freq)*(freq)])
    return sp.lti([1, decay], p_den_x)

#Problem 1
(t_1, x_1) = sp.impulse(trans_x(1.5, 0.5), None, np.linspace(0, 50, 3001))
py.figure(1)
py.plot(t_1, x_1, 'g')
py.xlabel('t')
py.ylabel('x')
py.grid(True)
py.title('Forced Damping Oscillator with Decay = 0.5')
py.savefig('1.png', dpi = 1000)
```

The plot for the above response $x(t)$ is as follows:



3.2 P2: Solving the above problem with a smaller decay

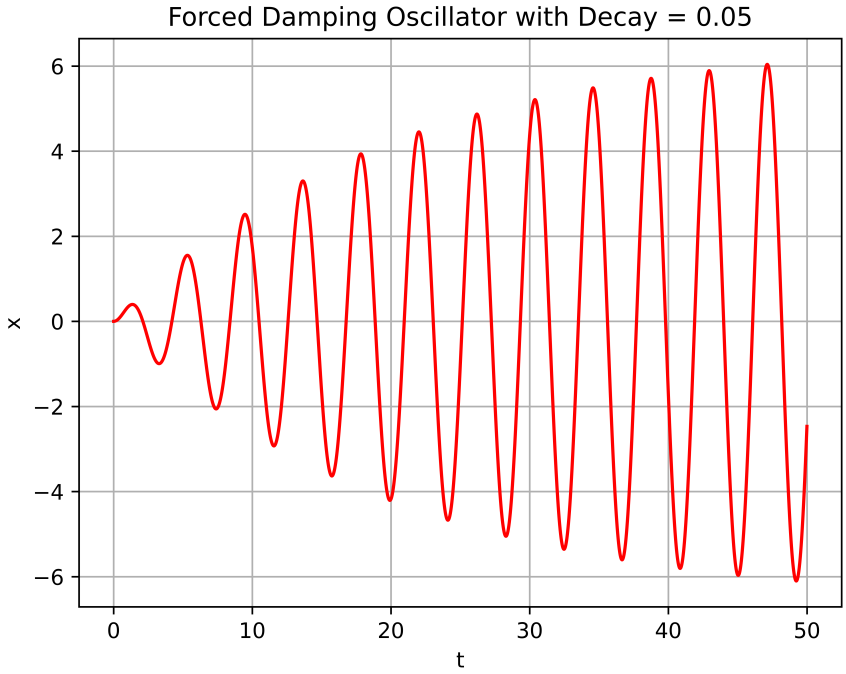
We will now change the decay in the previous problem from 0.5 to 0.05 that is,

$$f(t) = \cos(1.5t)e^{-0.05t}u_o(t)$$

The corresponding Laplace transform of $f(t)$ will be,

$$F(s) = \frac{s + 0.05}{(s + 0.05)^2 + 2.25}$$

Following similar procedure as was done in previous problem, we get the following plot for the $x(t)$ response:



The code for the computation is as follows

```
#Problem 2
(t_2, x_2) = sp.impulse(trans_x(1.5, 0.05), None, np.linspace(0, 50, 3001))
py.figure(2)
py.plot(t_2, x_2, 'r')
py.xlabel('t')
py.ylabel('x')
py.grid(True)
py.title('Forced Damping Oscillator with Decay = 0.05')
py.savefig('2.png', dpi = 1000)
```

where $X(s)$ generating function is defined earlier in the code.

3.3 P3: Analysing the variation in the frequency of $f(t)$ input

Firstly, we will obtain the system transfer function $\frac{X(s)}{F(s)}$ from the Laplace equation,

$$s^2X(s) + 2.25X(s) = F(s)$$

We get it as,

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 2.25}$$

We can create a function to return the system transfer function as follows:

```
#defining the system transfer function X(s)/F(s)
def trans_sys():
    return sp.lti([1.], [1., 0, 2.25])
```

We have to plot the response of the system when the frequency of the input function $f(t)$ is varied between 1.4 to 1.6 with a step of 0.05, keeping the damping factor constant at 0.05.

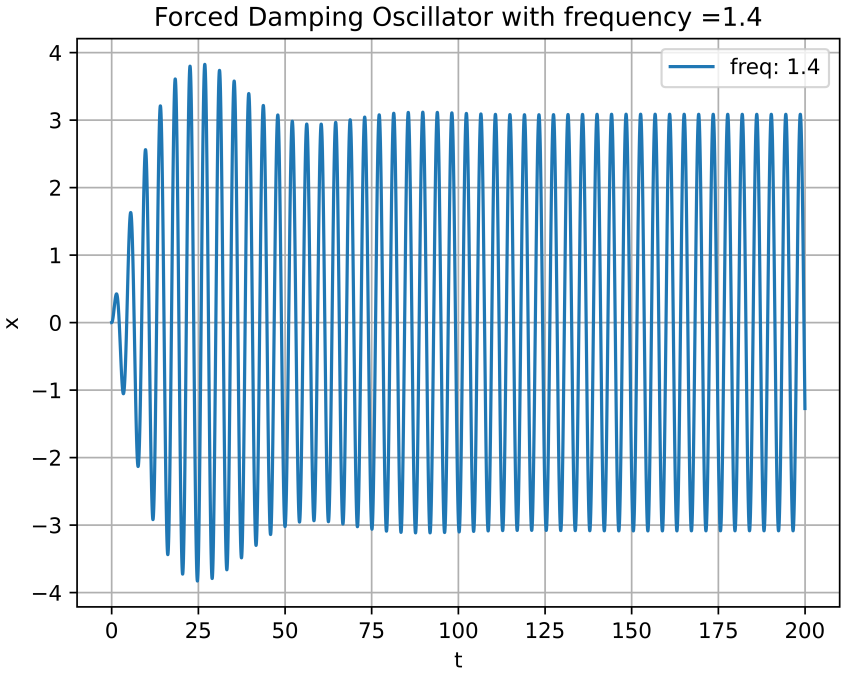
$$f(t) = \cos(wt)e^{-0.05t}u_o(t)$$

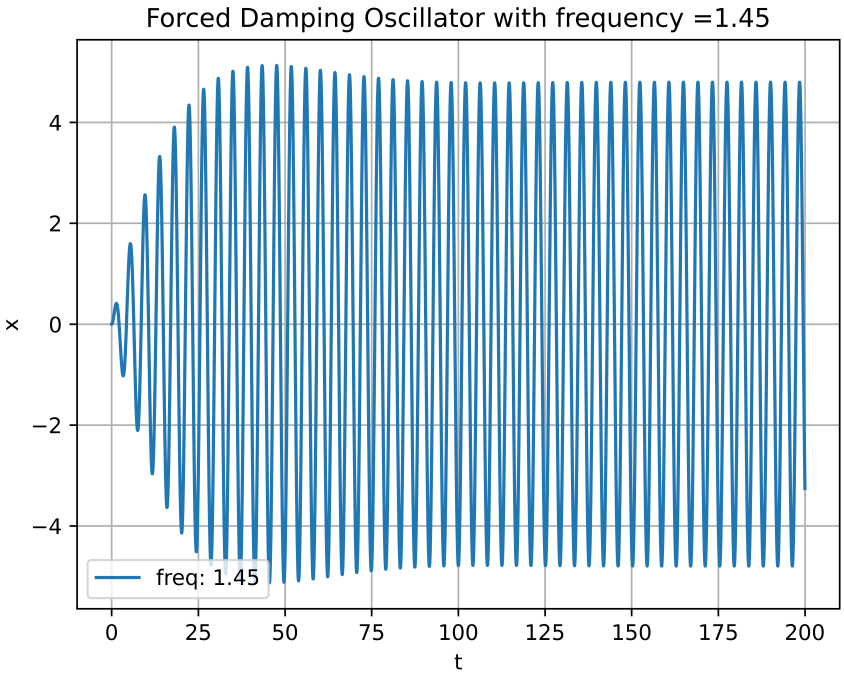
$$w = (1.4, 1.45, 1.5, 1.55, 1.6)$$

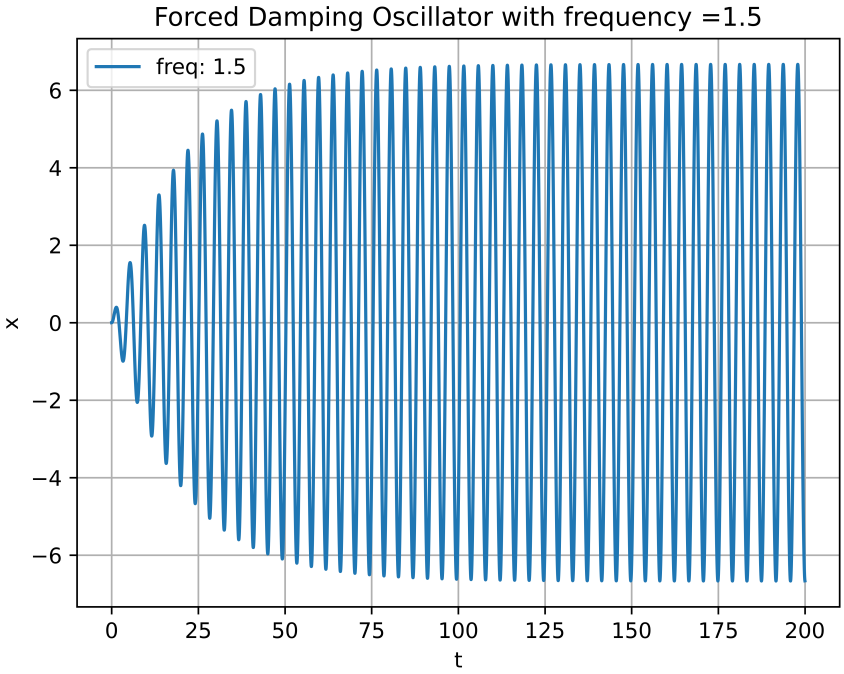
We can use the python function **lsim** to simulate the output for given input function and transfer function and run a loop for each frequency case. The code looks as follows:

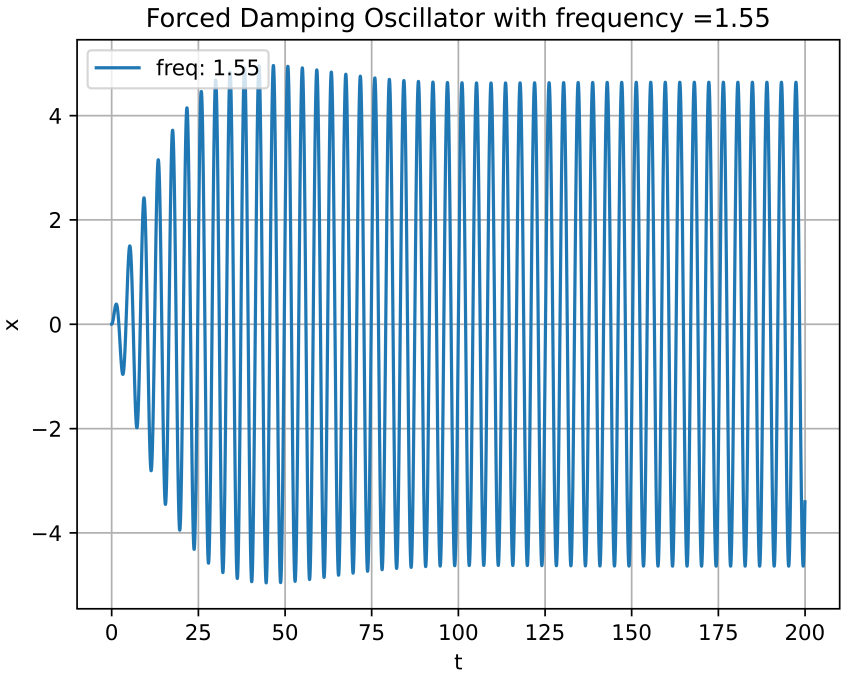
```
#Problem 3
fig_index = 3
t_3 = np.linspace(0, 200, 9001)
freq_sample = np.arange(1.4, 1.65, 0.05)

for index in freq_sample:
    py.figure(fig_index)
    f_t = np.cos(index*t_3)*np.exp(-0.05*t_3)
    (t, x_3, svec) = sp.lsim(trans_sys(), f_t, t_3)
    py.plot(t, x_3, label = 'freq: ' + str(index))
    py.xlabel('t')
    py.ylabel('x')
    py.legend()
    py.grid(True)
    py.title('Forced Damping Oscillator with frequency =' + str(index))
    py.savefig(str(fig_index) + '.png', dpi = 1000)
    fig_index += 1
```

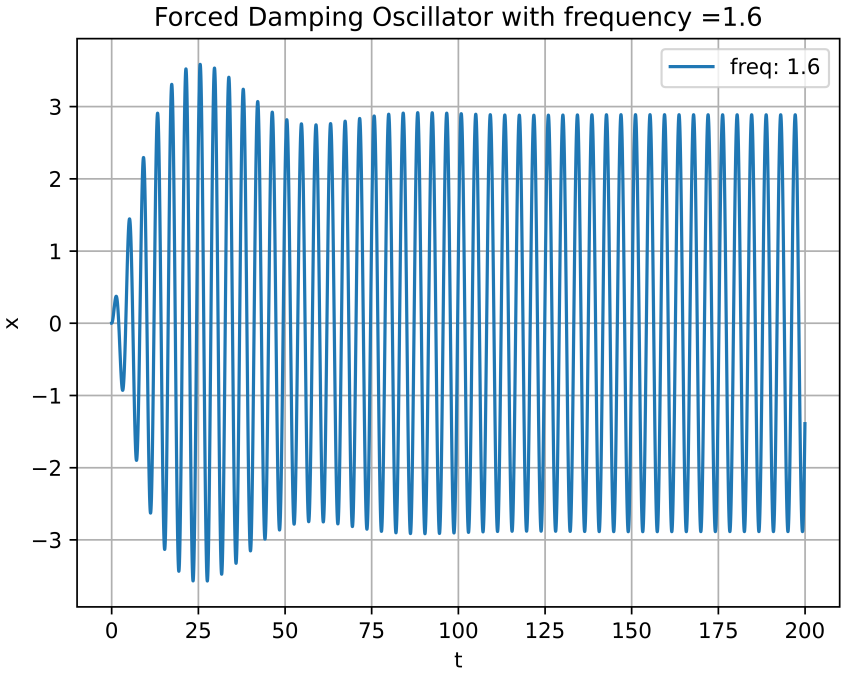
The plots look as follows:


Forced Damping Oscillator with frequency =1.4
freq: 1.4
x
t
4
3
2
1
0
−1
−2
−3
−4
0
25
50
75
100
125
150
175
200


Forced Damping Oscillator with frequency =1.45
freq: 1.45
x
t
4
2
0
−2
−4
0
25
50
75
100
125
150
175
200


Forced Damping Oscillator with frequency =1.5
freq: 1.5
x
t
6
4
2
0
−2
−4
−6
0
25
50
75
100
125
150
175
200


Forced Damping Oscillator with frequency =1.55
freq: 1.55
x
t
4
2
0
−2
−4
0
25
50
75
100
125
150
175
200



The observation from the plots are as follows:

- For frequencies less than 1.5, the overall amplitude of the plot overshoots and then settles at a constant value. Also, higher the frequency, higher is the settling amplitude and small overshoot and lesser the frequency, more is the extent of overshoot and settling amplitude is less.
- For frequencies greater than 1.5, the overall amplitude of the plot overshoots and then settles at a steady value. But, in here, higher the frequency, lower is the settling amplitude and high overshoot, and lesser the frequency, there is low overshoot and higher settling amplitude.
- At frequency of 1.5, the overall amplitude traces an exponentially settling plot and the settling amplitude is the highest for this frequency.

3.4 P4: Solving a Coupled Spring Problem

We have been given a set of two equations denoting the coupled spring motion in x and y direction as follows,

$$\frac{d^2x(t)}{dt^2} + x(t) - y(t) = 0$$

$$\frac{d^2y(t)}{dt^2} + 2(y(t) - x(t)) = 0$$

The initial conditions are; $x(0) = 1$, $x'(0) = y(0) = y'(0) = 0$. Solving the above equation by substituting y(t) from 2^{nd} to 1^{st} equation yields,

$$\frac{d^4x(t)}{dt^4} + 3\frac{d^2x(t)}{dt^2} = 0$$

From the first equation, we can say that, $\frac{d^2x(0)}{dt^2} = -1$. Now using the differential property of Laplace transform as mentioned earlier, in the 4^{th} order equation, we obtain the following Laplace equation,

$$s^4X(s) - s^3 + s + 3s^2X(s) - 3s = 0$$

Solving this we get,

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

From the 2^{nd} order y(t) equation, in Laplace domain, we obtain,

$$s^2Y(s) = 2X(s) - 2Y(s)$$

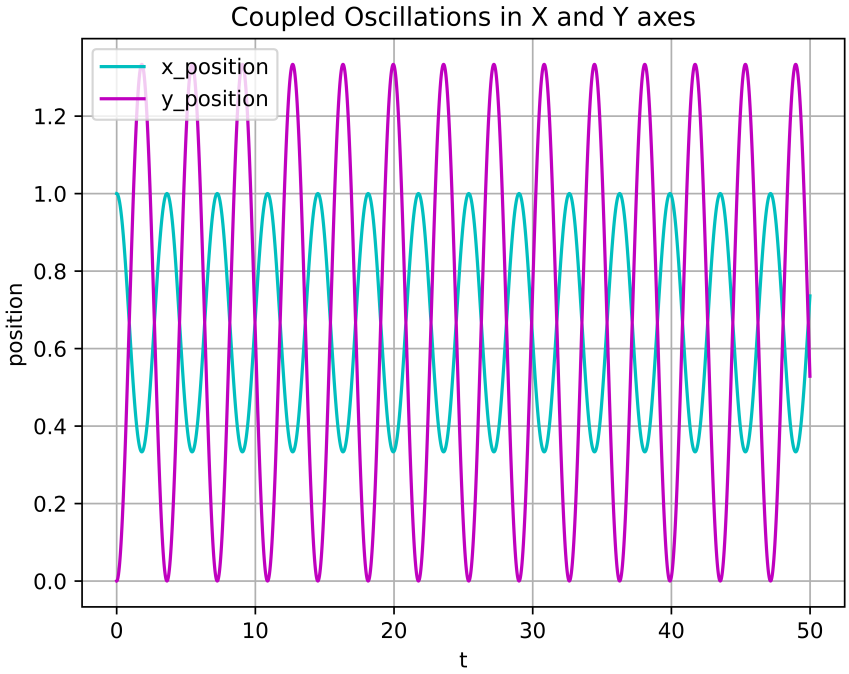
Substituting $X(s)$ in the equation, we obtain,

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

We can use the python function **impulse** to get the time domain functions $x(t)$ and $y(t)$. The code to construct the Laplace functions $X(s)$ and $Y(s)$, to find the time based response and to plot the same is as follows:

```
#Problem 4
X_4 = sp.lti([1, 0, 2], [1, 0, 3, 0])
Y_4 = sp.lti([2], [1, 0, 3, 0])

(t, x_4) = sp.impulse(X_4, None, np.linspace(0, 50, 3001))
(t, y_4) = sp.impulse(Y_4, None, np.linspace(0, 50, 3001))

py.figure(8)
py.plot(t, x_4, 'c', label = 'x_position')
py.plot(t, y_4, 'm', label = 'y_position')
py.xlabel('t')
py.ylabel('position')
py.legend()
py.grid(True)
py.title('Coupled Oscillations in X and Y axes')
py.savefig('8.png', dpi = 1000)
```

The plot for X and Y positions looks as follows:



3.5 P5: Obtaining Bode Plots for a Two Port Network

We are given a series R, L, C circuit in which input is applied at the inlet of R and output is taken across C. We are given $R = 100\Omega$, $L = 1\mu H$, $C = 1\mu F$. In steady state, for a sinusoidal input, the transfer function will look as follows:

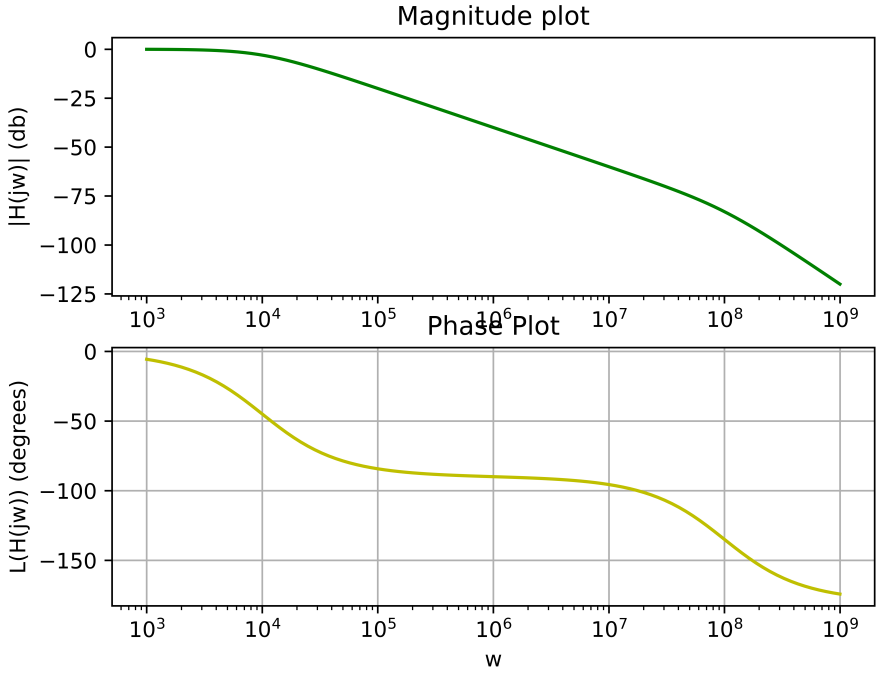
$$\frac{V_o}{V_i}(j\omega) = \frac{1}{1 + j\omega CR + (j\omega)^2 LC}$$

Bode Plots comprises of the magnitude and phase plots of the above transfer function. In the magnitude plot, the y-axis is in dB and x-axis is in log-scale. In the phase plot also the x-axis is in log-scale. The plots are computed using **bode** function in python, the code is as follows:

```
#Problem 5
R = 100
L = 1e-6
C = 1e-6
H_ckt = sp.lti([1], [L*C, R*C, 1])
(w_ckt, s_ckt, phi_ckt) = H_ckt.bode()
py.figure(9)
py.subplot(2,1,1)
py.semilogx(w_ckt, s_ckt, 'g')
py.ylabel('|H(jw)| (db)')
```

```
py.title('Magnitude plot')
py.subplot(2,1,2)
py.semilogx(w_ckt, phi_ckt, 'y')
py.xlabel('w')
py.ylabel(r'L(H(jw)) (degrees)')
py.title('Phase Plot')
py.grid(True)
py.savefig('9.png', dpi = 1000)
```

The Bode plot is as follows:



3.6 P6: Solving the above equation for a given $\mathbf{V}_i(t)$

We are given the input voltage $V_i(t)$ as follows,

$$V_i(t) = \cos(10^3 t)u(t) - \cos(10^6 t)u(t)$$

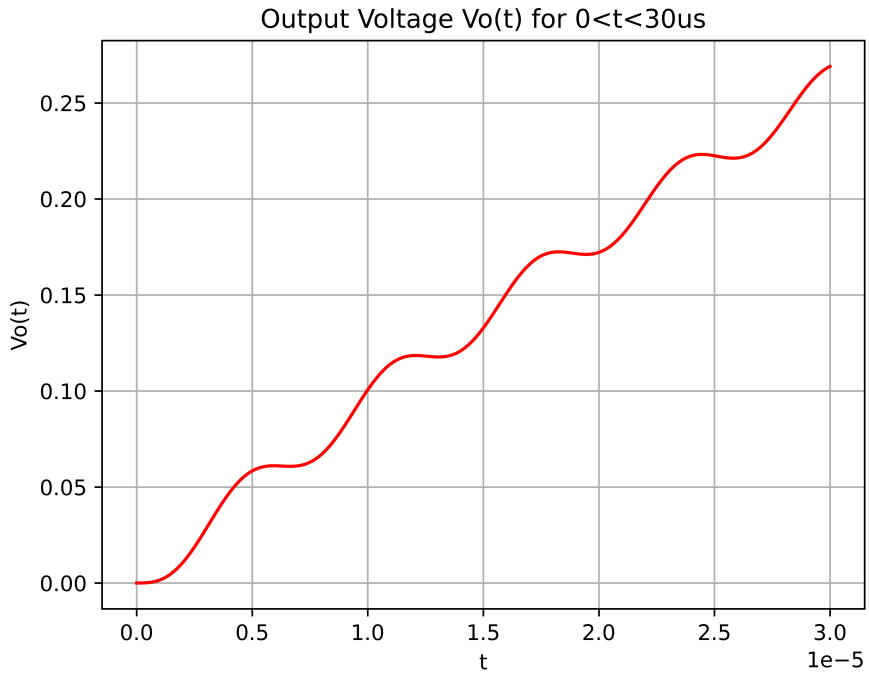
The transfer function for the system is given as follows,

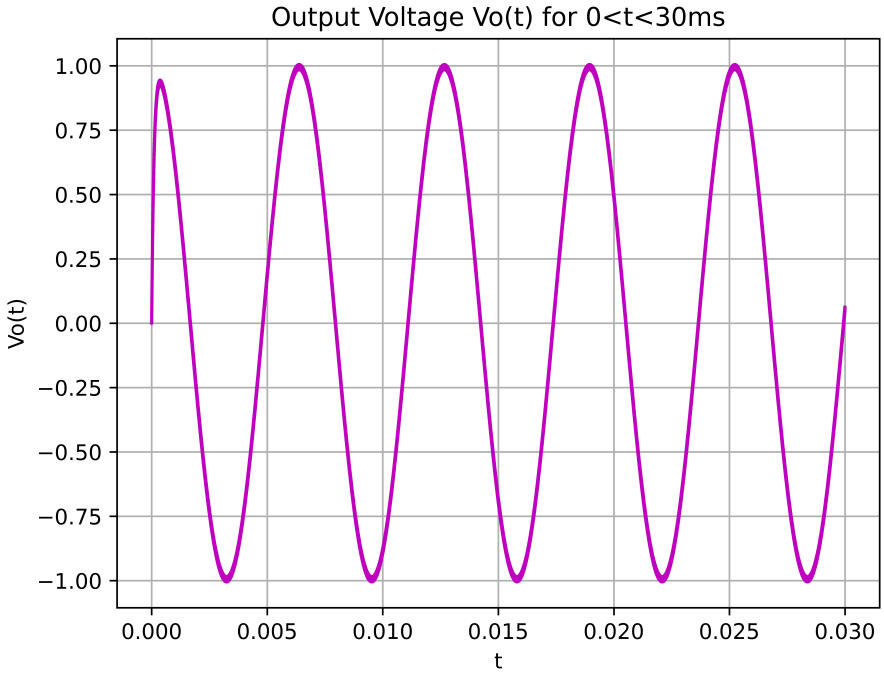
$$\frac{V_o}{V_i}(s) = \frac{1}{1 + sCR + s^2LC}$$

Using the python function **lsim**, we can obtain the response for the given input and transfer function. The code to find the response and to plot the same is as follows:

```
#Problem 6
#for 0<t<30us case
t_ckt1 = np.arange(0, 301e-7, 1e-7)
vi_t1 = np.cos(1e3*t_ckt1) - np.cos(1e6*t_ckt1)
(t_ckt1, vo_t1, svec) = sp.lsim(H_ckt, vi_t1, t_ckt1)
py.figure(10)
py.plot(t_ckt1, vo_t1, 'r')
py.xlabel('t')
py.ylabel('Vo(t)')
py.title('Output Voltage Vo(t) for 0<t<30us')
py.grid(True)
py.savefig('10.png', dpi = 1000)

#for long t (ms) variation
t_ckt2 = np.arange(0, 30e-3, 1e-7)
vi_t2 = np.cos(1e3*t_ckt2) - np.cos(1e6*t_ckt2)
(t_ckt2, vo_t2, svec) = sp.lsim(H_ckt, vi_t2, t_ckt2)
py.figure(11)
py.plot(t_ckt2, vo_t2, 'm')
py.xlabel('t')
py.ylabel('Vo(t)')
py.title('Output Voltage Vo(t) for 0<t<30ms')
py.grid(True)
py.savefig('11.png', dpi = 1000)
```

In here, we have taken two intervals in t to plot the response. One is given as $t < 30\mu s$ and other one is $t < 30ms$. To analyse the fast response, the distance between two t samples is taken as 10^-7. The plots look as follows:


Output Voltage Vo(t) for 0<t<30us
Vo(t)
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e−5
t


Output Voltage Vo(t) for 0<t<30ms
Vo(t)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.000
0.005
0.010
0.015
0.020
0.025
0.030
t

The conclusion for the plots is as follows:

- For the $t < 30\mu s$ analysis, the slow oscillating response (one having $\omega = 10^3$) is almost constant over the interval, and the fast oscillating response (one having $\omega = 10^6$) undergoes changes. Hence we are able to see a step kind of plot, which only captures the oscillations of the fast response with some shift by the almost constant slow response.
- For the $t < 30ms$ analysis, the slow oscillating response (one having $\omega = 10^3$) oscillates appreciably over the interval along with the fast oscillating response (one having $\omega = 10^6$). Hence we are able to see an overall sinusoidal response (by the slow portion) with a rippling effect (by the fast portion).