Assignment - 6

Ramanan S - EE18B145

March 10, 2020

1 Introduction

In this assignment, we will look at how to analyse "Linear Time-invariant Systems" with numerical tools in Python. We will be analysing the systems in both the Laplace domain and in the time domain.

We will be using tools from the 'scipy' library for the purpose of analysing various LTI systems.

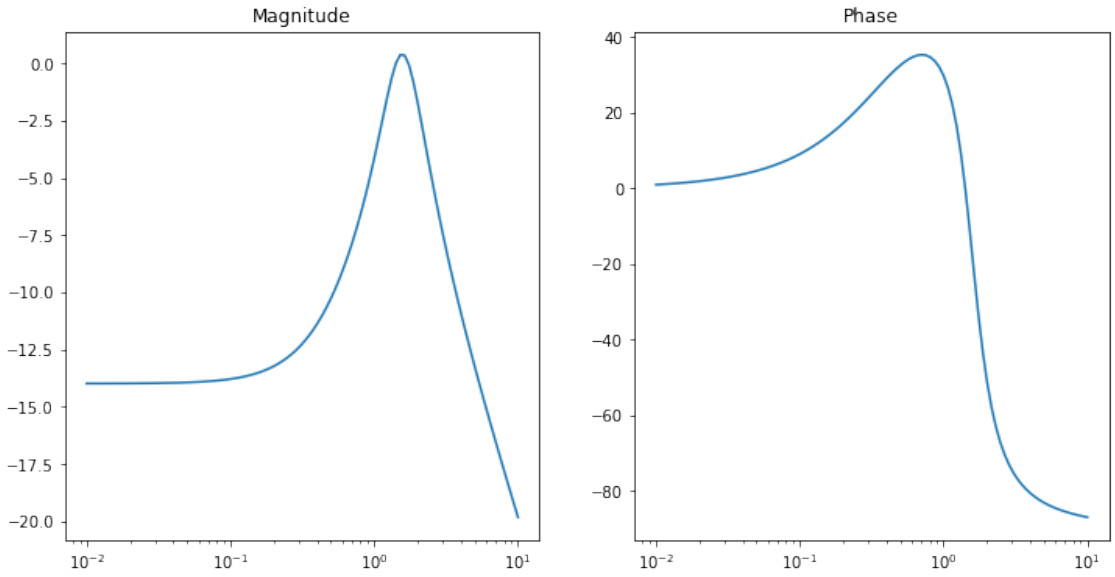
2 Procedure

2.1 Part - 0

In this part we will be using various functions from the library *scipy.signal* and make ourselves familiarize with those functions.

[1]:
```
#Importing required libraries
import scipy.signal as sp
from pylab import *
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#Plotting the magnitude and phase variation of the given function with frequency
H = sp.lti([1,0.5], [1,1,2.5])
w,s,phi = H.bode()
f = figure(figsize=(12,6))

#Magnitude Vs freq.
ax = f.add_subplot(121)
ax.semilogx(w,s)
ax.set_title('Magnitude')

#Phase Vs freq.
ax2 = f.add_subplot(122)
ax2.semilogx(w,phi)
ax2.set_title('Phase')
show()
```

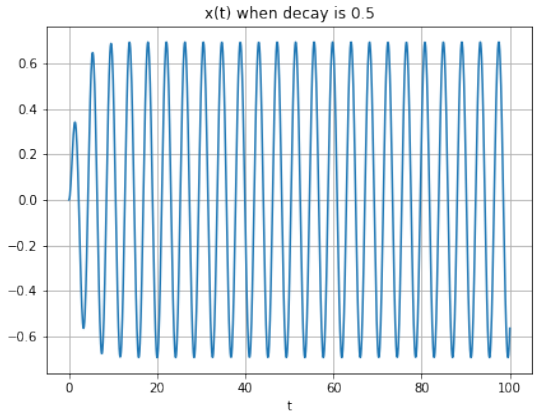


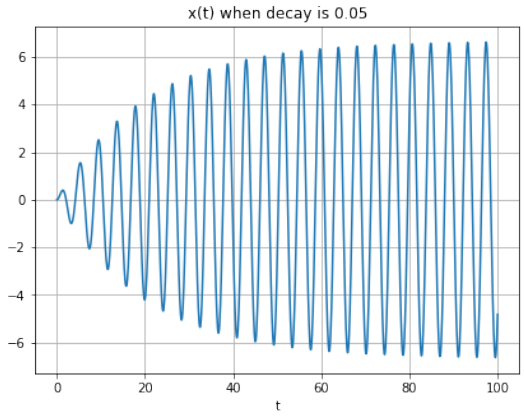
2.2 Part - 1&2

In this part we will solve the spring system quation for different decay rates.

[3]:
```
#Function which solves the laplace equation for a given 'decay rate' and␣
 ↪'frequency'
def solve(d, w):
    M = [1,0,2.25]
    tmp = polymul([1,d],[1,d])
    tmp = polyadd(tmp,w*w)
    a = polymul(M,tmp)
    X = sp.lti([1,d],a)
    #Converts the X in 'laplace' domain to 'time' domain
    t,x=sp.impulse(X,None,linspace(0,100,1001))

    return t, x

#Plotting the solution of spring system for different decay rates keeping the␣
 ↪frequency constant
f = figure(figsize=(15,5))
ax = f.add_subplot(121)
ax2 = f.add_subplot(122)

#Plot when decay=0.5
t, x = solve(0.5,1.5)
ax.plot(t,x)
ax.set_title('x(t) when decay is 0.5')
ax.set_xlabel('t')
```

```
ax.grid()

#Plot when decay=0.05
t, x = solve(0.05,1.5)
ax2.plot(t,x)
ax2.set_title('x(t) when decay is 0.05')
ax2.set_xlabel('t')
ax2.grid()

show()
```





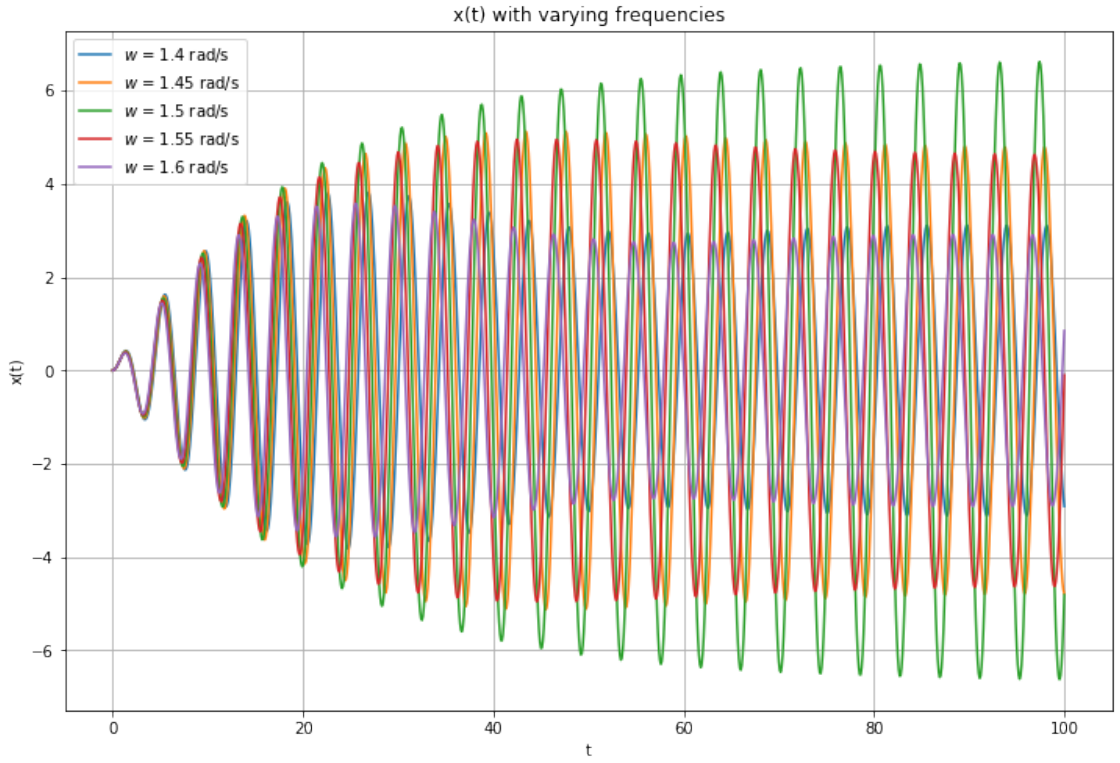
We can see that, if the 'deacay rate' is smaller, then the amplitude of oscillations shoots up for the given period of time. Also from the graph,we can see that, $x(t)$ attains steady state faster, if the the decay rate is higher.

2.3 Part - 3

In this part, we will be plotting $x(t)$ for different values of the frequencies.

[4]:
```
#Plotting the solution x(t) for various frequencies keeping the decay rate␣
 ↪constant
f2 = figure(figsize=(12,8))
ax = f2.add_subplot(111)

#Freq. Range - [1.4,1.6]
for w in arange(1.4,1.6,0.05):
    t,x = solve(0.05,w)
    ax.plot(t, x, label="$w$ = %g rad/s" % (w))

ax.set_title('x(t) with varying frequencies')
ax.set_xlabel('t')
ax.set_ylabel('x(t)')
ax.legend()
```

```
ax.grid()
```



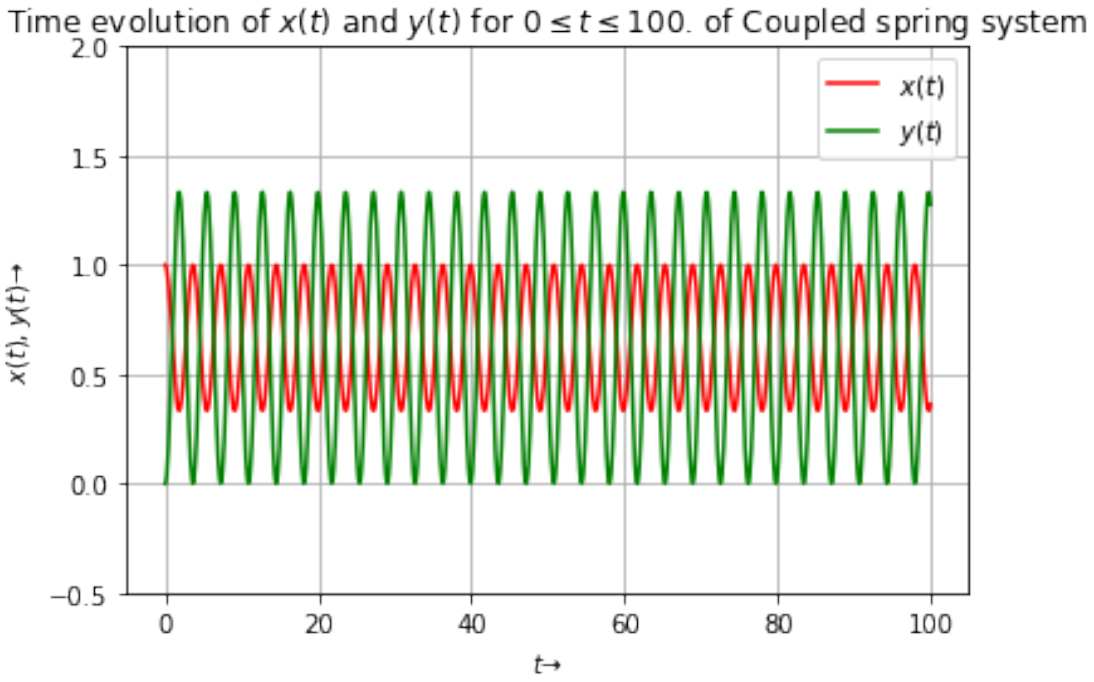
We can see that the amplitude of oscillations is the highest for $w = 1.5 rads^{-}1$. We can conclude that $1.5 rads^{-}1$ is the resonant frequency of this system.

2.4 Part - 4

In this part, we will solve x and y for the given coupled equation and plot them to see how both of them varies with time.

[5]:
```
#Function which transforms the given Hs into h(t) - (i.e.) time domain
#Function takes the numerator and denominator of Hs function as arguments
def solve_2(num, den):
    num = poly1d(num)
    den = poly1d(den)
    Hs = sp.lti(num, den)

    #converting H(s) to h(t)
    t, h = sp.impulse(Hs, None, linspace(0, 100, 1000))
    return t, h

#Solving for the coupled equation which contains both 'x' and 'y'
t1, x = solve_2([1, 0, 2], [1, 0, 3, 0])
```

```
t2, y = solve_2([2], [1, 0, 3, 0])

#Plotting 'x(t)' and 'y(t)' in the same graph
plot(t1, x, 'r', label="$x(t)$")
plot(t2, y, 'g', label="$y(t)$")
title("Time evolution of $x(t)$ and $y(t)$ for $0 \leq t \leq 100$. of Coupled␣
 ↪spring system ")
xlabel(r"$t \to $")
ylabel(r"$x(t),y(t) \to $")
ylim((-0.5, 2))
legend()
grid()
show()
```



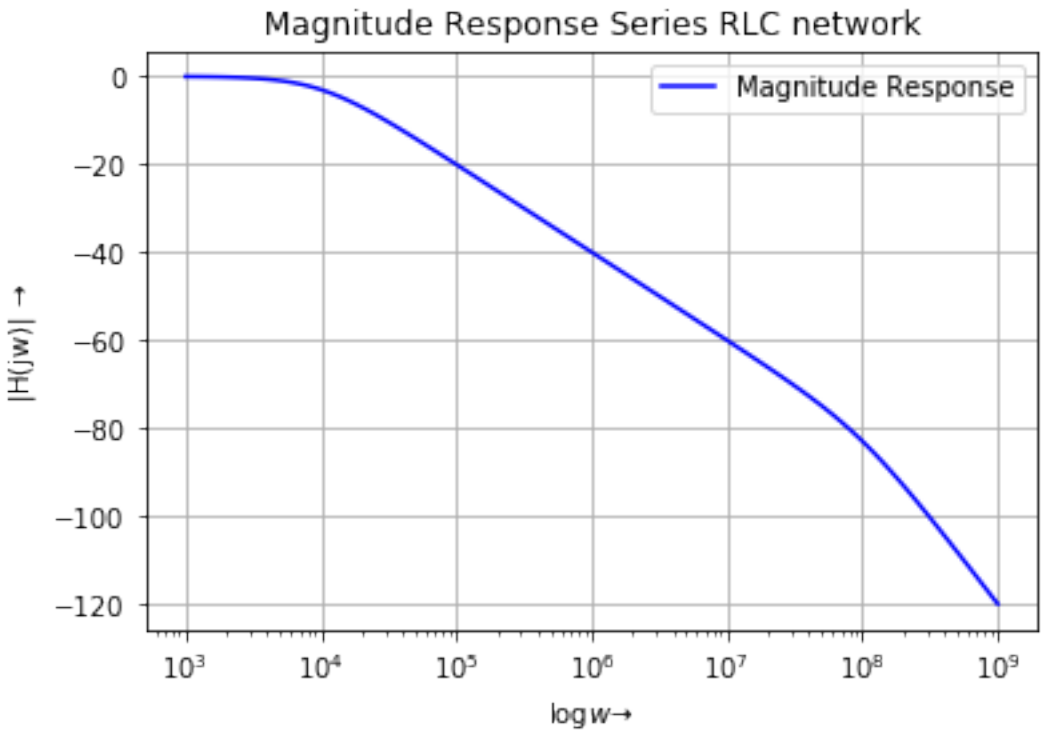
We can observe from the graph that, both $x(t)$ and $y(t)$ satisfies the initial conditions and are out of phase with each other by 180^0.

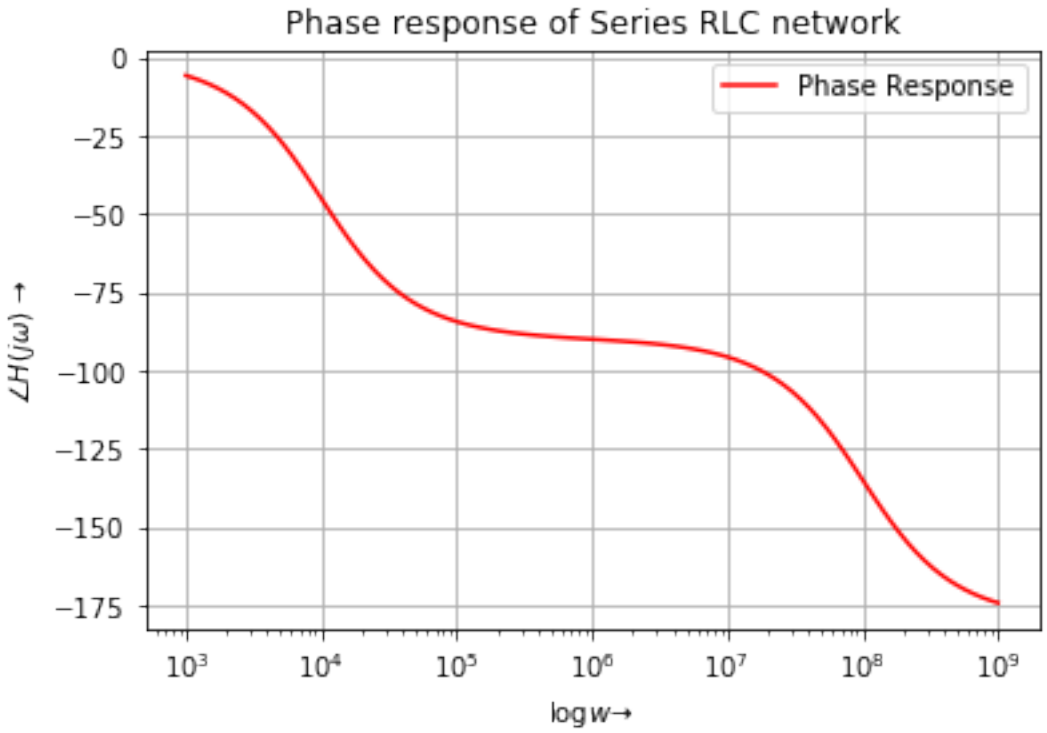
2.5 Part - 5

In this part, we will find the transfer function of the given RLC circuit and then plot both the 'Magnitude' and 'Phase' response of the system.

[6]:
```
#Function which solves the transfer function of the given RLC circuit
def RLC_circuit(R=100, L=10**-6, C=10**-6):
    num = poly1d([1])
```

```
    den = poly1d([L*C,R*C,1])

    Hs = sp.lti(num, den)
    w, mag, phi = Hs.bode()
    return w, mag, phi, Hs

w, mag, phi, H = RLC_circuit()

# plot Magnitude Response
semilogx(w, mag, 'b', label="Magnitude Response")
legend()
title("Magnitude Response Series RLC network")
xlabel(r"$ \log w \to $")
ylabel(r"|H(jw)| $\to $")
grid()
show()

# Plot of phase response
semilogx(w, phi, 'r', label="Phase Response")
legend()
title("Phase response of Series RLC network")
xlabel(r"$ \log w \to $")
ylabel(r"$ \angle H(j\omega)$ $\to $")
grid()
show()
```





The given RLC circuit acts like a Low-pass filter (i.e) gain is close to zero for $w >= 10^4 rads^-1$. We can see from the equation of transfer function that it totally has 2 poles and each pole adds 90^0 to the phase plot. The system is unconditionally stable, as phase is never greater than 180^0.

2.6 Part - 6

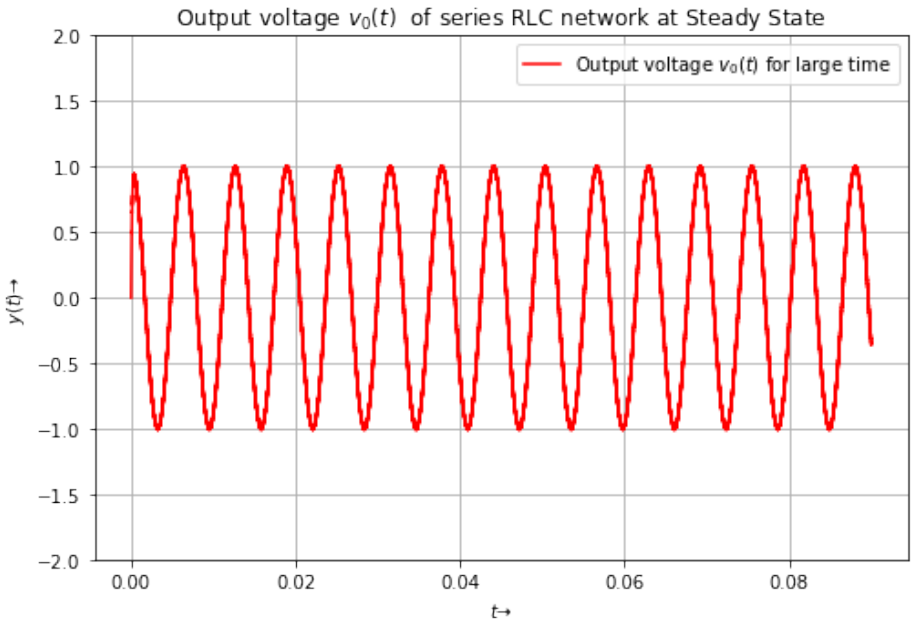
In this part, we are going to find the output V_0 for the given input V_i using the transfer function of the circuit.

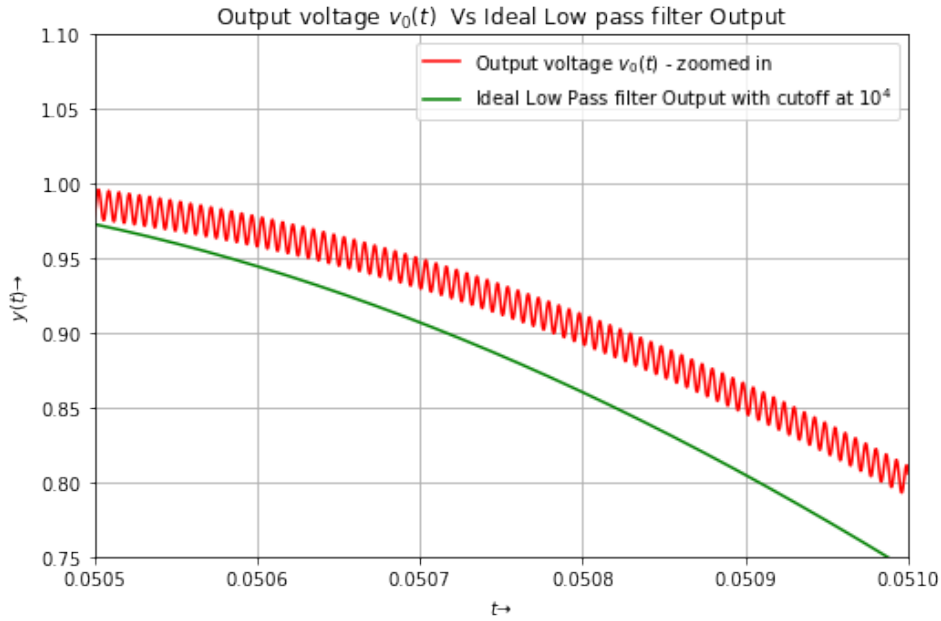
$$V_0 = V_i * H$$

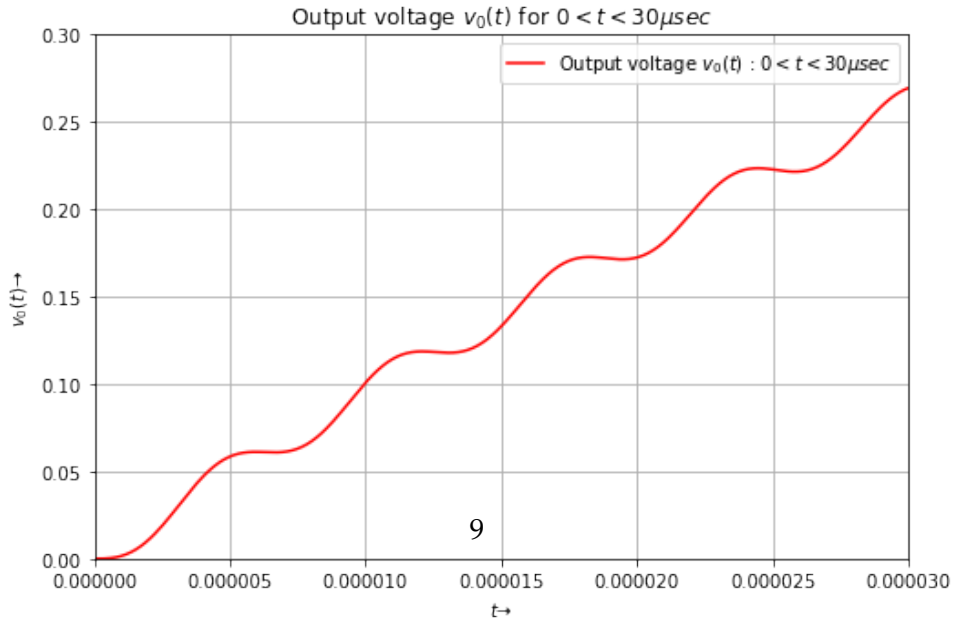
where H is the transfer function of the given RLC circuit.

[7]:
```
#Defining the time for which values to be noted
t = linspace(0, 90*pow(10, -3), pow(10, 6))
#Defining the input - Vi
vi = cos(t*pow(10, 3))-cos(t*pow(10, 6))

#Convoluting Vi with H(s) and getting the output Vo
t, vo, svec = sp.lsim(H, vi, t)

#As a ideal low-pass filter lets only if freq. <= 10**4 rad/s
vo_ideal = cos(1e3*t)
```

```
f3 = figure(figsize=(8,18))
ax1 = f3.add_subplot(311)
ax2 = f3.add_subplot(312)
ax3 = f3.add_subplot(313)

#Plot of the output Vo Vs t - for large values of t
ax1.plot(t, vo, 'r', label="Output voltage $v_0(t)$ for large time")
ax1.legend()
ax1.set_ylim(-2, 2)
ax1.set_title(r"Output voltage $v_0(t)$  of series RLC network at Steady State")
ax1.set_xlabel(r"$ t \to $")
ax1.set_ylabel(r"$ y(t) \to $")
ax1.grid()

#Ideal Plot of Vo Vs the real plot of Vo and time
ax2.plot(t, vo, 'r', label="Output voltage $v_0(t)$ - zoomed in ")
ax2.plot(t, vo_ideal, 'g', label="Ideal Low Pass filter Output with cutoff at␣
 ↪$10^4$")
ax2.set_xlim(0.0505, 0.051)
ax2.set_ylim(0.75, 1.1)
ax2.legend()
ax2.set_title(r"Output voltage $v_0(t)$  Vs Ideal Low pass filter Output")
ax2.set_xlabel(r"$ t \to $")
ax2.set_ylabel(r"$ y(t) \to $")
ax2.grid()

#Plot of the output Vo Vs t - for small values of t
ax3.plot(t, vo, 'r', label="Output voltage $v_0(t)$ : $0<t<30\mu sec$")
ax3.legend()
ax3.set_title(r"Output voltage $v_0(t)$ for $0<t<30\mu sec$")
ax3.set_xlim(0, 3e-5)
ax3.set_ylim(-1e-5, 0.3)
ax3.set_xlabel(r"$ t \to $")
ax3.set_ylabel(r"$ v_0(t) \to $")
ax3.grid()
f3.show()
```


Output voltage $v_0(t)$ of series RLC network at Steady State
Output voltage $v_0(t)$ for large time
$y(t)\rightarrow$
$t\rightarrow$


Output voltage $v_0(t)$ Vs Ideal Low pass filter Output
Output voltage $v_0(t)$ - zoomed in
Ideal Low Pass filter Output with cutoff at 10^4
$y(t)\rightarrow$
$t\rightarrow$


Output voltage $v_0(t)$ for $0 < t < 30\mu sec$
Output voltage $v_0(t)$: $0 < t < 30\mu sec$
$v_0(t)\rightarrow$
$t\rightarrow$

As we observe the plot and the circuit that we know it is a Low pass filter with bandwidth $0 < \omega < 10^4$.

So when the circuit will only pass input with frequencies which are in range of bandwidth only. But since its not a ideal low pass filter as its gain doesn't drop abruptly at 10^4 rather gradual decrease which is observed from magnitude response plot.

So the output $V_o(t)$ will be mainly of $cos(10^3 t)$ with higher frequencies riding over it in long term response i.e Steady state solution.

This behaviour is observed in the plot that,the $v_o(t) \approx \cos(10^3 t)$ with higher frequencies riding over it for large time.

The curve is very flickery or not smooth because of the high frequency components only. In the second plot, we've zoomed in large enough to observe those high-frequency components.

However, the oscillations due to higher frequencies are small as their gain is much less.

3 Conclusion

So to conclude we analysed a new way to find the solution of continuous-time LTI systems using Laplace transform with help of SciPy signals toolbox implemented in Python and got ourselves familiarised with solving the differential equations using laplace transform instead of doing tedious time domain analysis.