The Laplace Transform

Name: Sankalp Saoji (EE16B063)

Date: 19/03/2018

Abstract

In this week, we looked at how to analyse mechanical examples of "Linear Time-invariant Systems" with numerical tools in Python. LTI systems are what Electrical Engineers spend most of their time thinking about - linear circuit analysis or communication channels for example.

1 INTRODUCTION

All the problems in the assignment are in "continuous time" and will use Laplace Transforms. Python has a Signals toolbox which is very useful and complete. In this assignment, we explored the following commands:

```
Command               Description               Example
1)poly1d              Defines polynomials       p = poly1d([1, 2, 3]
2)polyadd             Adds the polynomials      q = polyadd([1, 2], [9, 5, 4])
3)polymul             Multiplies polynomials    r = polymul([1,2],[9,5,4])
```

For importing signal toolbox, we do,

```
import scipy.signal as sp
```

Following are the various commnds that we explored in scipy.signal module,

```
Command                         Description                     Example
1)sp.lti                        Defines a transfer function     H=sp.lti([1,2,1],[1,2,1,1])


     [Declaring three variables like,
                                                                  subplot(2,1,1)
                                                                  semilogx(w,S)
                                                                  subplot(2,1,2)
                                                                  semilogx(w,phi)


2)sp.impulse                    Computes the impulse response   t,x=sp.impulse(H,None,
                                of the transfer function        linspace(0,10,101)
                                                                [Giving a particular range of t using
                                                                linspace helps us in getting the
                                                                selected x values only]

3)sp.lsim                       Calculates the convolution      t=linspace(0,10,101)
                                of given functions              u=sin(t)
                                                                t,y,svec=sp.lsim(H,u,t)
                                                                We get the convolution of u and t.
```

2 INITIALIZATION

2.1 Importing Libraries and Modules :

```
# Importing libraries and modules

import numpy
from pylab import *
import math
import matplotlib.pyplot as plt
import scipy.signal as sp
```

3 QUESTIONS

3.1 Getting x(t) from its impulse response and plotting:

f(t) = cos(1.5t)$e^{-0.5t}u_0$(t)
Laplace Transform of f(t) is,

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

We had to solve for the time response of a spring satisfying,

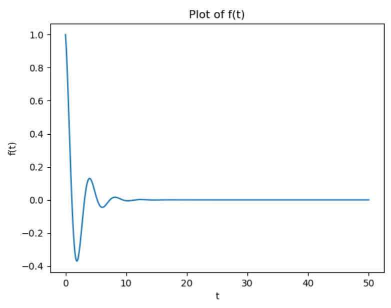
$\frac{d^2x}{dt^2}$ + 2.25x = f(t)

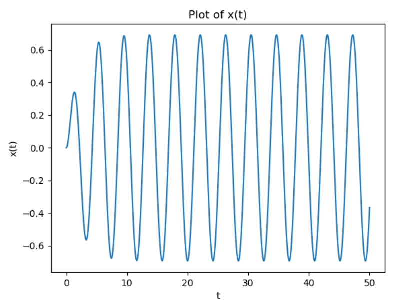
x(0) = 0, $\frac{dx}{dt}$ = 0
0<t<50

```
#Obtaining f(t) and x(t)

#Obtaining f(t)

F_num = poly1d([1,0.5])
F_denom = poly1d([1,1,2.5])
F = sp.lti(F_num, F_denom)
t_f, f = sp.impulse(F, None, linspace(0,50,1001))

#Obtaining x(t)

X_num = F_num
X_denom = polymul([1,0,2.25], F_denom)
X = sp.lti(X_num, X_denom)
t_x, x = sp.impulse(X, None, linspace(0,50,1001))
x[0] = 0
```

In the above code, I have obtained f(t) and x(t) from their impulse responses using 'sp.impulse'.

```
#Plotting f(t) and x(t)

#Plotting f(t)

 figure(0)
 plt.plot(t_f,f)
 plt.title('Plot of f(t)')
 plt.xlabel('t')
 plt.ylabel('f(t)')
 plt.show()

#Plotting x(t)

 figure(1)
 plt.plot(t_x,x)
 plt.title('Plot of x(t)')
 plt.xlabel('t')
 plt.ylabel('x(t)')
 plt.show()
```





These are the plots I obtained.

3.2 Getting a changed x(t) from its impulse response and plotting:

Now,

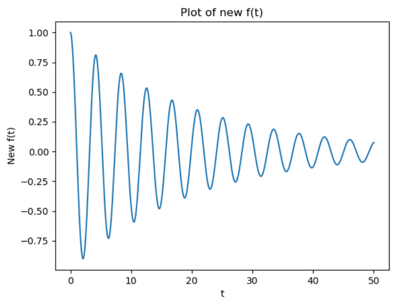
$$f(t) = \cos(1.5t)e^{-0.05t}u_0(t)$$

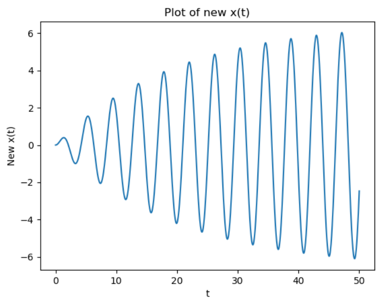
We have a lesser decay this time. So, we had to solve for the response of the spring.

```
#Obtaining new f(t) and new x(t)

#Obtaining new f(t)

F_new_num = poly1d([1,0.05])
F_new_denom = poly1d([1,0.1,2.2525])
F_new = sp.lti(F_new_num, F_new_denom)
t_new_f, f_new = sp.impulse(F_new, None, linspace(0,50,1001))

#Obtaining new x(t)

X_new_num = F_new_num
X_new_denom = polymul([1,0,2.25], F_new_denom)
X_new = sp.lti(X_new_num, X_new_denom)
t_new_x, x_new = sp.impulse(X_new, None, linspace(0,50,1001))
x_new[0] = 0
```

In the above code, I have obtained the new f(t) and x(t) from their impulse responses using 'sp.impulse'.

```
#Plotting new f(t) and new x(t)

#Plotting new f(t)

figure(2)
plt.plot(t_new_f,f_new)
plt.title('Plot of new f(t)')
plt.xlabel('t')
plt.ylabel('New f(t)')
plt.show()

#Plotting new x(t)

figure(3)
plt.plot(t_new_x,x_new)
plt.title('Plot of new x(t)')
plt.xlabel('t')
plt.ylabel('New x(t)')
plt.show()
```



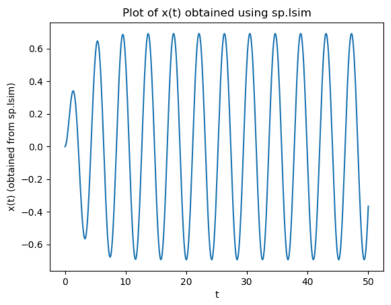


3.3 Varying the frequency of the cosine and plotting:

In this the problem, we conidered f (t) is the input, and x(t) is the output. We had to obtain the system transfer function X(s)/F(s) using signal.lsim to simulate the problem. We had to vary the frequency of the cosine in f (t) from 1.4 to 1.6 in steps of 0.05 keeping the exponent as exp(−0.05t) and plot the resulting responses.

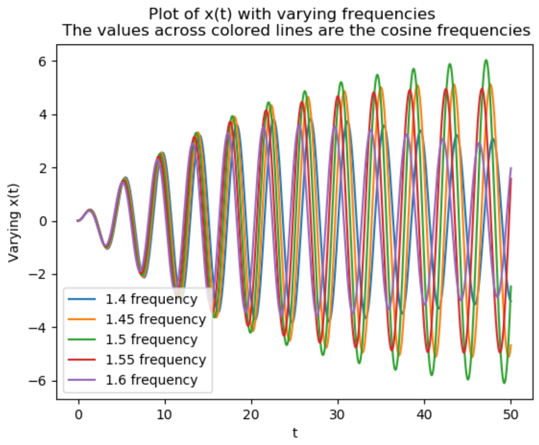
```
#Obtaining x(t) from sp.lsim

#Obtaining x(t) from sp.lsim

H = sp.lti(polymul(X_num,F_denom), polymul(X_denom, F_num))
t_in = linspace(0,50,1001)
t_sim, x_sim, vec = sp.lsim(H, f, t_in)

#Plotting x(t) obtained from sp.lsim

figure(4)
plt.plot(t_sim, x_sim)
plt.title('Plot of x(t) obtained using sp.lsim')
plt.xlabel('t')
plt.ylabel('x(t) (obtained from sp.lsim)')
plt.show()
```

I have obtained x(t) from sp.lsim so as to show the accuracy of this method. As we see from below plot, x(t) obtained is very accurate.



Now, we had to plot x(t) in a for loop with varying frequency each time. So, we do,

```
#Plotting in for loop

omega = linspace(1.4, 1.6, 5)

for i in omega:
        F_new_num2 = poly1d([1,0.05])
        F_new_denom2 = poly1d([1,0.1,i*i + 0.0025])
        F_new2 = sp.lti(F_new_num2, F_new_denom2)
        t_new_f2, f_new2 = sp.impulse(F_new2, None, linspace(0,50,1001))
        t = linspace(0,50,1001)
        t_new_sim, x_sim2, vec_sim = sp.lsim(H,f_new2, t)
        figure(5)
        plt.plot(t_new_sim, x_sim2, label = '%s frequency' %i)
        plt.title('Plot of x(t) with varying frequencies \n
The values across colored lines are the cosine frequencies')
        plt.xlabel('t')
        plt.ylabel('Varying x(t)')
        plt.legend()
plt.show()
```



As we see from the graph, we get the peak at 1.5 frequency. So, 1.5 is the resonant frequency of the function.

3.4 Solving the coupled spring problem:

Given equations are,

$$\ddot{x} + (x - y) = 0$$

$$\ddot{y} + 2(y - x) = 0$$

where the initial condition is,

$$x(0) = 1, \dot{x}(0) = y(0) = \dot{y}(0) = 0$$

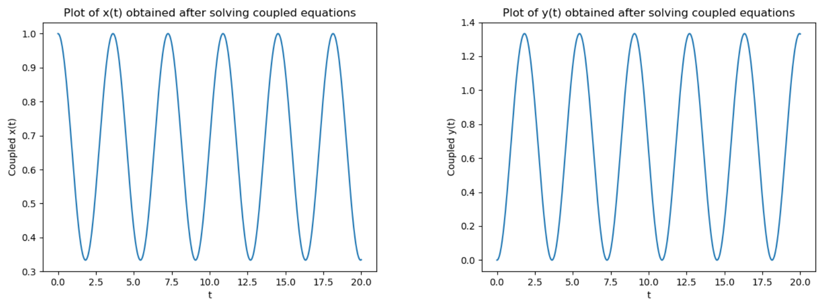
After substituting and solving, we get a fourth order equation.

```
#Solving the coupled equations and getting x(t) and y(t)

X_coup_num = poly1d([1, 0, 2])
X_coup_denom = poly1d([1,0,3,0])
X_coup = sp.lti(X_coup_num, X_coup_denom)
Y_coup_num = poly1d([2])
Y_coup_denom = poly1d([1,0,3,0])
Y_coup = sp.lti(Y_coup_num, Y_coup_denom)
t_coup_x, x_coup = sp.impulse(X_coup, None, linspace(0,20,1001))
t_coup_y, y_coup = sp.impulse(Y_coup, None, linspace(0,20,1001))
```

I have solved for x(t) and y(t) as above.

```
#Plotting x(t) and y(t) from the coupled equations

figure(6)
```

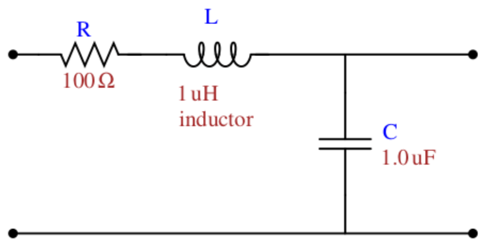
```
plt.plot(t_coup_x, x_coup)
plt.title('Plot of x(t) obtained after solving coupled equations')
plt.xlabel('t')
plt.ylabel('Coupled x(t)')
plt.show()

figure(7)
plt.plot(t_coup_y, y_coup)
plt.title('Plot of y(t) obtained after solving coupled equations')
plt.xlabel('t')
plt.ylabel('Coupled y(t)')
plt.show()
```



Above are the graphs I obtained.

3.5 Obtaining the magintude and phase plots of the steady state response of a given network :

Below is the two port network whose steady state function we wish to find,



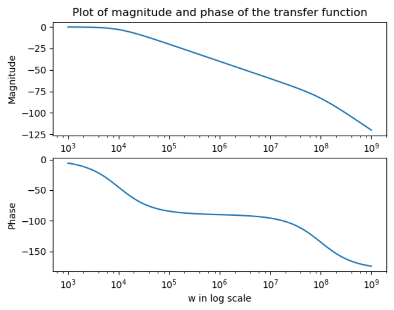
After solving, I got the transfer function as follows,

```
#Solving for the steady state response of the two port network

T = sp.lti(poly1d([1]), poly1d([10**-12, 10**-4, 1]))
omega_transfer, mag, phi = T.bode()
```

T.bode() helps separating T and get 'magnitude' and 'phase'. Here, python gives us the magnitude in decibels.

```
#Plotting the magnitude and phase of the transfer function

figure(8)
plt.subplot(2,1,1)
plt.semilogx(omega_transfer, mag)
plt.title('Plot of magnitude and phase of the transfer function')
plt.xlabel('w in log scale')
plt.ylabel('Magnitude')
plt.subplot( 2,1,2)
plt.semilogx(omega_transfer, phi)
plt.xlabel('w in log scale')
plt.ylabel('Phase')
plt.show()
```

Below are the obtained plots as a subplot,



Magnitude and phase plot of the transfer function of the two port network.

3.6 Getting vo(t) when an input has been declared:

We had given the following input signal to the above two-port network,

v_i(t) = cos(10^3t)u(t) - cos(10^6t)u(t)

We needed to capture the fast variation and so, the time step chosen was smaller than 10 $^{-6}$. We must see the slow time and so, I simulated till about 10 msec. Since the network is resistive, the current at t = 0^- decayed to zero, which gives us our initial condition. Below is the way I have obtained v_o(t).

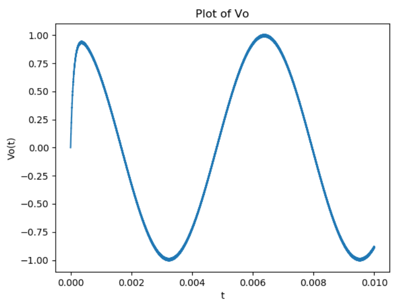
```
#Obtaining vo of the function

t_vo = linspace(0,0.01,100001)
```

```
vi = cos((10**3)*t_vo) - cos((10**6)*t_vo)
t_transfer, vo, vec_transfer = sp.lsim(T, vi, t_vo)
```

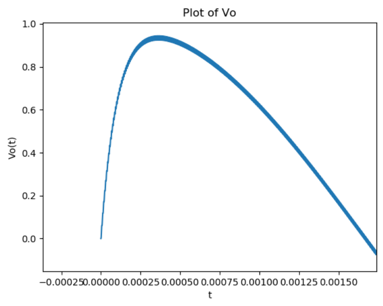
Shown below is the procedure for plotting v_out,

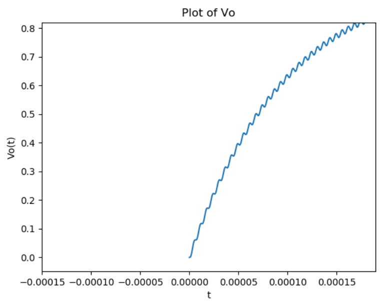
```
#Plotting vo

figure(10)
plt.plot(t_transfer, vo)
plt.title('Plot of Vo')
plt.xlabel('t')
plt.ylabel('Vo(t)')
plt.show()
```

Following are the obtained plots,



Below are the sections of the above plot, we can clearly see the ripple in the plots,





As we see from the above graphs, the grapg of v$_o$(t) is composed of two parts,

1) The natural response of the system which continues upto about 30μs (this response comes into picture due to the sudden application of input v$_i(t)$ to the steady system) and,

2) The cosine part which we obtain due to filtering from the low pass filter.

There are a lot of ripples present due to the non-ideal low pass filter as shown in the graph.