Assignment7

Saurabh Vaishampayan: EP17B028

April 06 2020

1 Introduction

The assignment involves using sympy module in Python to solve for transfer functions from a given input to output in a circuit network. We solve for two networks: Second order Low pass and High pass filter and analyse the step response, impulse response, output for a sum of high and low frequency sinusoidal input and exponentially damped sinusoidal inputs of different frequencies.
We implement the following tasks in the program:

1. Define functions that generate symbolic matrices and solve them for the given circuit with inputs as values of components. Return the matrix, output vector in symbolic form.
2. Extract the required node voltage. Pass this on to a new function which simplifies the symbolic transfer function, extracts the coefficients for numerator and denominator polynomials and returns the lti transfer function to be used using signal processing toolbox.
3. Find impulse response and step response(divide transfer function by s) by scipy.signal.impulse. Plot corresponding Bode plots and analyse the outputs.
4. Find outputs to inputs of:
   - Sum of sinusoids of low and high frequencies.
   - Exponentially decaying sinusoids of low and high frequencies

   We use scipy.signal.lsim for this

2 Importing libraries, defining functions for symbolic matrices

2.1 Code

```
import numpy as np
import scipy.signal as sp
import sympy
import matplotlib.pyplot as plt

sympy.init_session
sympy.init_printing(use_latex='mathjax')

s = sympy.symbols('s')

```

```
def lowpass(R1,R2,C1,C2,G,Vi):
    A = sympy.Matrix([[0,0,1,-1/G],[-1/(1+s*R2*C2),1,0,0],[0,-G,G,1],[-1/R1-1/R2-s*C1,1/R2,0,s*C1]])
    b = sympy.Matrix([0,0,0,Vi/R1])
    v = (A**-1)*b
    return A, b, v

def highpass(R1,R2,C1,C2,G,Vi):
    A = sympy.Matrix([[0,0,1,-1/G],[s*C2*R2/(1+s*C2*R2),-1,0,0],[0,G,-G,-1],[1+1/(s*C1*R1),1/(s*C1*R2),0,-1/(s*C1*R1)]])
    b = sympy.Matrix([0,0,0,Vi])
    v = (A**-1)*b
    return A, b, v
```

3 Function to extract lti transfer function from sympy expression

3.1 Code

```
def lti_transfer_func(H):
    H = sympy.simplify(H)
    n, d = sympy.fraction(H)

    num = sympy.Poly(n, s).all_coeffs()
    den = sympy.Poly(d, s).all_coeffs()

    num, den = [float(r) for r in num], [float(k) for k in den]

    Transf = sp.lti(num, den)

    return Transf
```

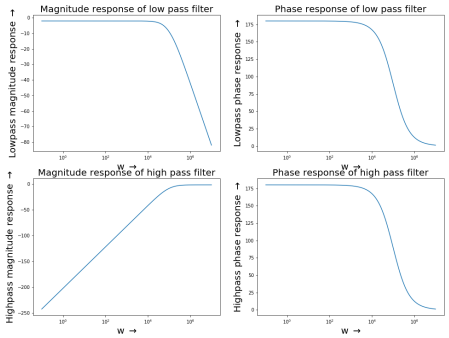
4 Bode plots of impulse response for LPF and HPF

4.1 Code

```
#Impulse response(Substitute Vi = 1, since Laplace domain)
A_lp, b_lp, v_lp = lowpass(1e4,1e4,1e-9,1e-9,1.586,1)
A_hp, b_hp, v_hp = highpass(1e4,1e4,1e-9,1e-9,1.586,1)

Vo_lp = v_lp[3]
Vo_hp = v_hp[3]


#Magnitude and phase response of Lowpass and Highpass filters

H1 = lti_transfer_func(Vo_lp)
w1 = np.logspace(-1,7,801)
w1, S1, phi1 = H1.bode(w=w1)

H2 = lti_transfer_func(Vo_hp)
w2 = np.logspace(-1,7,801)
w2, S2, phi2 = H2.bode(w=w2)

fig,ax = plt.subplots(2,2,sharex=False, figsize=(16,12))


ax[0][0].plot(w1,S1)
ax[0][0].set_xscale('log')
```

```
ax[0][0].set_xlabel('w '+r'$\rightarrow$', size=20)
ax[0][0].set_ylabel('Lowpass magnitude response '+r'$\rightarrow$', size=20)
ax[0][0].set_title('Magnitude response of low pass filter', size=20)

ax[0][1].plot(w1,phi1)
ax[0][1].set_xscale('log')
ax[0][1].set_xlabel('w '+r'$\rightarrow$', size=20)
ax[0][1].set_ylabel('Lowpass phase response '+r'$\rightarrow$', size=20)
ax[0][1].set_title('Phase response of low pass filter', size=20)

ax[1][0].plot(w2,S2)
ax[1][0].set_xscale('log')
ax[1][0].set_xlabel('w '+r'$\rightarrow$', size=20)
ax[1][0].set_ylabel('Highpass magnitude response '+r'$\rightarrow$', size=20)
ax[1][0].set_title('Magnitude response of high pass filter', size=20)

ax[1][1].plot(w2,phi2)
ax[1][1].set_xscale('log')
ax[1][1].set_xlabel('w '+r'$\rightarrow$', size=20)
ax[1][1].set_ylabel('Highpass phase response '+r'$\rightarrow$', size=20)
ax[1][1].set_title('Phase response of high pass filter', size=20)

fig.savefig('bodeplot.png')
```

4.2 Plots



4.3 Inference

We observe that the filters are second order, owing to 180 degrees phase change(Note that it starts at 180 degrees as transfer function has negative sign). From the magnitude plots it seems that they are near critically damped.

5 Step response

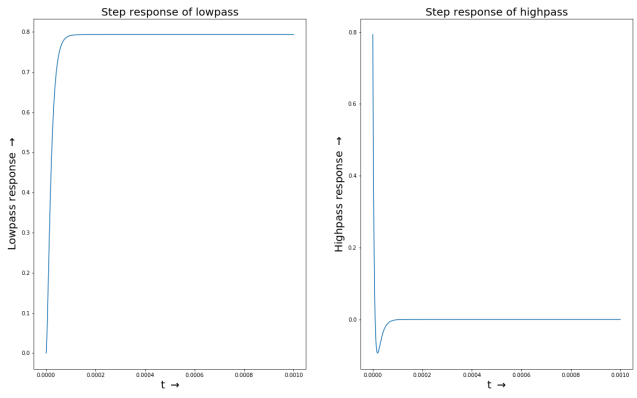
We divide the transfer function by s and extract the lti transfer function from it. Now we take impulse response to this function and plot it which is equivalent to step response.

5.1 Code

```
#Unit step response

```

```
Vo_lp_step = Vo_lp/s
Vo_hp_step = Vo_hp/s

H3 = lti_transfer_func(Vo_lp_step)
H4 = lti_transfer_func(Vo_hp_step)

t = np.linspace(0,0.001,2001)

t, x1 = sp.impulse(H3,X0=None,T=t)
t, x2 = sp.impulse(H4,X0=None,T=t)

fig2, ax2 = plt.subplots(1,2,figsize=(20,12))

ax2[0].plot(t,-x1)
ax2[0].set_xlabel('t '+r'$\rightarrow$', size=20)
ax2[0].set_ylabel('Lowpass response '+r'$\rightarrow$',size=20)
ax2[0].set_title('Step response of lowpass',size=20)

ax2[1].plot(t,x2)
ax2[1].set_xlabel('t '+r'$\rightarrow$', size=20)
ax2[1].set_ylabel('Highpass response '+r'$\rightarrow$',size=20)
ax2[1].set_title('Step response of highpass',size=20)
fig2.savefig('stepresponse.png')
```

5.2 Plots



5.3 Inference

Since the high pass filter rejects DC, we notice that the response approaches 0 as steady state value with a spike near t=0, which is due to presence of high frequencies as step is discontinuous. Low pass filter response exponentially approaches 0.7930 which is gain times the DC value.

6 Input as sum of sinusoids

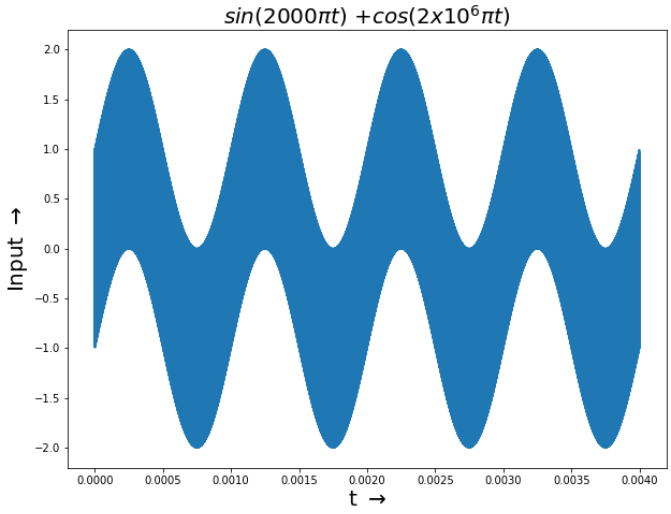
$$v_i(t) = (cos(2\pi \times 10^6 t) + sin(2000\pi t))u_0(t)$$

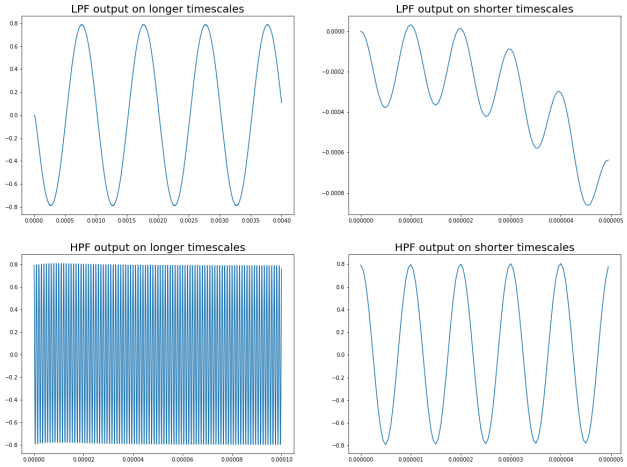
We use the lti transfer functions found earlier and use scipy.signal.lsim to obtain the output.

6.1 Code

```
#Sum of high and low frequency sinusoids
t = np.arange(0,0.004,5e-8)
f = np.sin(2000*np.pi*t)+np.cos(2e6*np.pi*t)

t, x3, svec = sp.lsim(H1, f, t, None)
t, x4, svec = sp.lsim(H2, f, t, None)

plt.figure(figsize=(10,7.5))
plt.plot(t,f)
plt.xlabel('t '+r'$\rightarrow$', size=20)
plt.ylabel('Input '+r'$\rightarrow$', size=20)
plt.title(r'$sin(2000 \pi t)$'+' +'+r'$cos(2x10^{6} \pi t)$', size=20)
plt.savefig('inputsumsin.png')

fig3, ax3 = plt.subplots(2,2,figsize=(20,15))

ax3[0][0].plot(t,x3)
ax3[0][0].set_title('LPF output on longer timescales',size=20)
ax3[0][1].plot(t[:100],x3[:100])
ax3[0][1].set_title('LPF output on shorter timescales',size=20)

ax3[1][0].plot(t[:2000],x4[:2000])
ax3[1][0].set_title('HPF output on longer timescales',size=20)
ax3[1][1].plot(t[:100],x4[:100])
ax3[1][1].set_title('HPF output on shorter timescales',size=20)

fig3.savefig('q2.png')
```

6.2 Plots





6.3 Inference

We notice that for LPF, at longer timescales, the output is the 1KHz sinuoid while a tiny variation corresponding to 1MHz can be observed at shorter timescales.
For HPF, at longer timescales, a tiny variation in peak amplitude to the 1MHz cosine can be observed. This is because filter has attenuated the 1KHz sinusoid. At shorter timescales one identifies the 1MHz signal cleanly

7 Damped sinusoids

We pass the following two damped sinusoidal inputs:

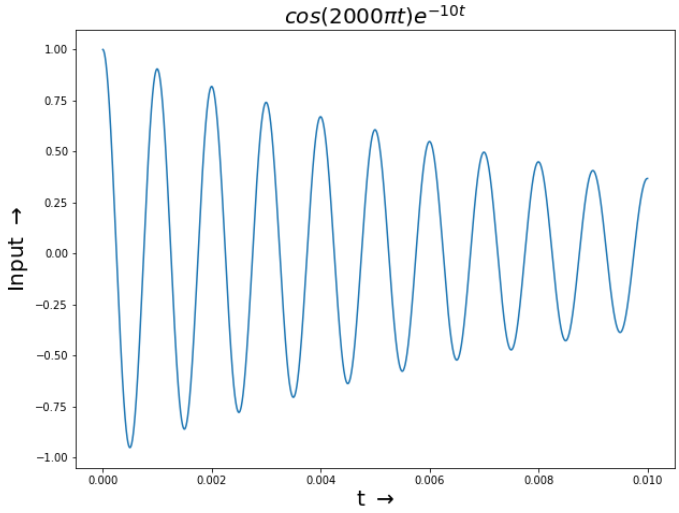
- $v_i(t) = cos(2\pi \times 10^3 t)e^{-100t}$
- $v_i(t) = cos(2\pi \times 10^7 t)e^{-10^6 t}$

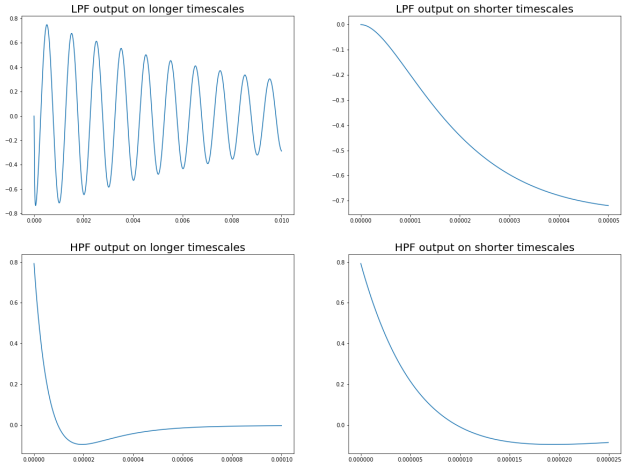
7.1 Code

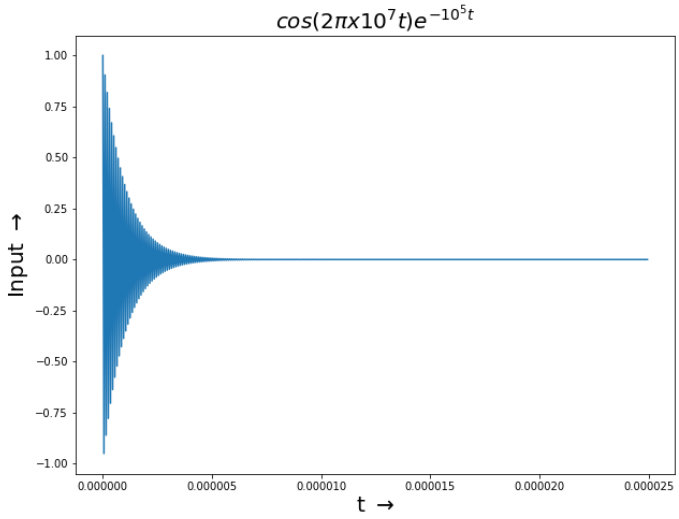
```
#Damped sinuoids
t = np.arange(0,1e-2,5e-8)
f1 = np.cos(2*np.pi*1e3*t)*np.exp(-100*t)
f2 = np.cos(2*np.pi*1e7*t)*np.exp(-1e6*t)

t, x5, svec = sp.lsim(H1, f1, t, None)
t, x6, svec = sp.lsim(H2, f1, t, None)

t, x7, svec = sp.lsim(H1, f2, t, None)
t, x8, svec = sp.lsim(H2, f2, t, None)

plt.figure(figsize=(10,7.5))
plt.plot(t,f1)
plt.xlabel('t '+r'$\rightarrow$', size=20)
plt.ylabel('Input '+r'$\rightarrow$', size=20)
plt.title(r'$cos(2000 \pi t)e^{-10t}$', size=20)
plt.savefig('lowfreqdamp.png')

fig4, ax4 = plt.subplots(2,2,figsize=(20,15))

ax4[0][0].plot(t,x5)
ax4[0][0].set_title('LPF output on longer timescales',size=20)
ax4[0][1].plot(t[:100],x5[:100])
ax4[0][1].set_title('LPF output on shorter timescales',size=20)

ax4[1][0].plot(t[:2000],x6[:2000])
ax4[1][0].set_title('HPF output on longer timescales',size=20)
ax4[1][1].plot(t[:100],x6[:100])
ax4[1][1].set_title('HPF output on shorter timescales',size=20)

fig4.savefig('lowfreqresp.png')

plt.figure(figsize=(10,7.5))
plt.plot(t[:1000],f2[:1000])
plt.xlabel('t '+r'$\rightarrow$', size=20)
plt.ylabel('Input '+r'$\rightarrow$', size=20)
plt.title(r'$cos(2 \pi x10^{7} t)e^{-10^{5}t}$', size=20)
plt.savefig('highfreqdamp.png')

fig5, ax5 = plt.subplots(2,2,figsize=(20,15))

ax5[0][0].plot(t,x7)
ax5[0][0].set_title('LPF output on longer timescales',size=20)
ax5[0][1].plot(t[:100],x7[:100])
ax5[0][1].set_title('LPF output on shorter timescales',size=20)

ax5[1][0].plot(t[:2000],x8[:2000])
ax5[1][0].set_title('HPF output on longer timescales',size=20)
ax5[1][1].plot(t[:100],x8[:100])
```

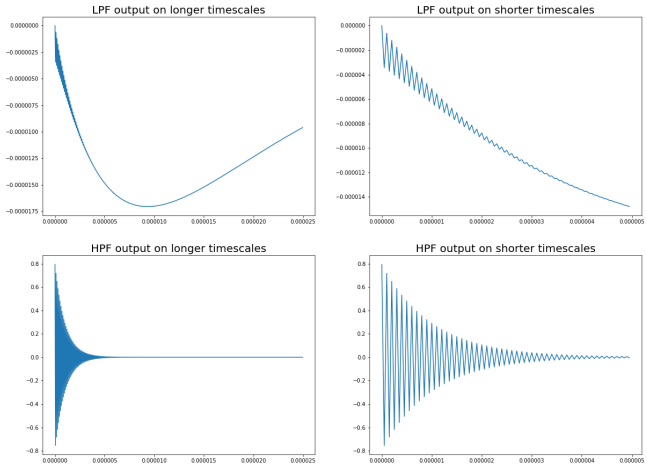
```
ax5[1][1].set_title('HPF output on shorter timescales',size=20)

fig5.savefig('responsehighfreq.png')
```

7.2 Plots









7.3 Inference

1. Low frequency damped sinusoid:
    - The low pass filter lets the sinusoid pass through without much attenuation. However the output also eventually decays since the input also exponentially decays
    - The high pass filter quickly attenuates the input. It is also much faster than the LPF as input frequency is much lower than cutoff frequency
2. High frequency damped sinusoid:
    - The lowpass filter takes much longer than the high pass filter to respond since the input frequency is much larger than cutoff frequency. It shows very small fluctutations

of the high sinusoidal frequency.

- The highpass filter lets the input pass almost like what it originally was, with a slight attenuation