EE2703: Applied Programming Lab
Assignment 7
Filter Simulation using Sympy

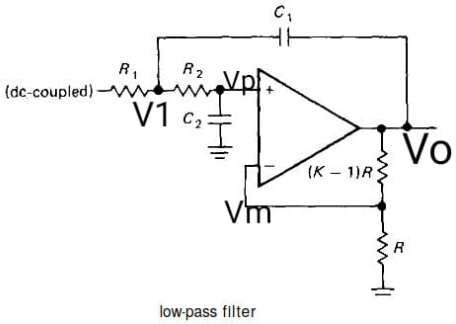
Arjun Menon Vadakkeveedu
EE18B104
Electrical Engineering, IIT Madras

April 7, 2020

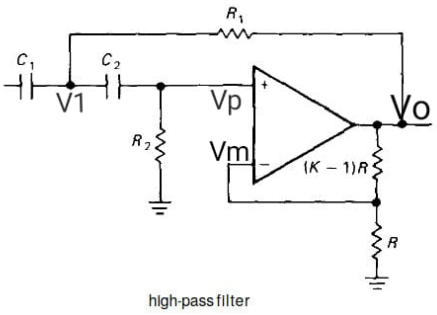
Contents

1 Introduction

In this program, the following filters are implemented using Sympy and Scipy.signal modules:



Low Pass Filter Circuit, source: Horowitz and Hill, *The Art of Electronics*



High Pass Filter Circuit, source: Horowitz and Hill, *The Art of Electronics*

A symbolic expression for the transfer function of the filters may be obtained by solving the following matrix equations in s-domain:
Low Pass Filter:

$$\begin{bmatrix} 0 & 0 & 1 & \frac{-1}{G} \\ \frac{-1}{1+sR2C2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ \frac{-1}{R1} - \frac{1}{R2} - sC1 & \frac{1}{R2} & 0 & sC1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{V_i}{R1} \end{bmatrix}$$

High Pass Filter:

$$\begin{bmatrix} 0 & 0 & G & -1 \\ 0 & -G & G & 1 \\ sC2R2 & -1 - sC2R2 & 0 & 0 \\ \frac{1}{R1} + sC1 + sC2 & -sC2 & 0 & \frac{-1}{R1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{V_i}{R1} \end{bmatrix}$$

Since the aim here is to find the impulse response of the filter, V_i is set to 1.

---

2 Code Structure

2.1 Important blocks used in the program

1: Symbolic Expression of Filter's Transfer Function

```
def lpf(R1, R2, C1, C2, G):
 A = sym.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 = sym.Matrix([0, 0, 0, 1/R1])
 V = A.inv()*B
 Vo = V[3]
 Vo = sym.simplify(Vo)
 num_H, den_H = sym.fraction(Vo)
 n_p = sym.Poly(num_H, s).all_coeffs()
 d_p = sym.Poly(den_H, s).all_coeffs()
 num = np.array(n_p, dtype = float)
 den = np.array(d_p, dtype = float)
 H = sp.lti(num, den)
 return (Vo, H)


def hpf(R1, R3, C1, C2, G):
 A = sym.Matrix([[0, 0, G, -1], [0, -G, G, 1],
    [s*C2*R3, -1 - s*C2*R3, 0, 0], [(1/R1 + s*C1 + s*C2), -s*C2, 0, -1/R1]])
 B = sym.Matrix([0, 0, 0, s*C1*1])
 V = A.inv()*B
 Vo = V[3]
 Vo = sym.simplify(Vo)
 num_H, den_H = sym.fraction(Vo)
 n_p = sym.Poly(num_H, s).all_coeffs()
 d_p = sym.Poly(den_H, s).all_coeffs()
 num = np.array(n_p, dtype = float)
 den = np.array(d_p, dtype = float)
 H = sp.lti(num, den)
 return (Vo, H)
```

2: Response of Filter to a Particular Input Signal

```
s = sym.symbols('s')
Vo_l, H_lpf = lpf(10e3, 10e3, 10e-12, 10e-12, G)
# H is the frequency domain impulse response
w_lpf, S_lpf, phi_lpf = H_lpf.bode()
t_lpf = np.arange(0, 2, 2e-3)
u_l1 = np.ones(t_lpf.shape)
# lsim determines the convolution of h = ilaplace(H) with causal signals u,
# here u is a heaviside fn and o/p is calculated for the first 20 units of t
t_lpf, y_l1, __ = sp.lsim(H_lpf, u_l1, t_lpf)
```

```
u_l2 = np.sin(2000*np.pi*t_lpf) + np.cos(2e6*np.pi*t_lpf)
t_lpf, y_l2, __ = sp.lsim(H_lpf, u_l2, t_lpf)
```

2.2 Miscellaneous Details

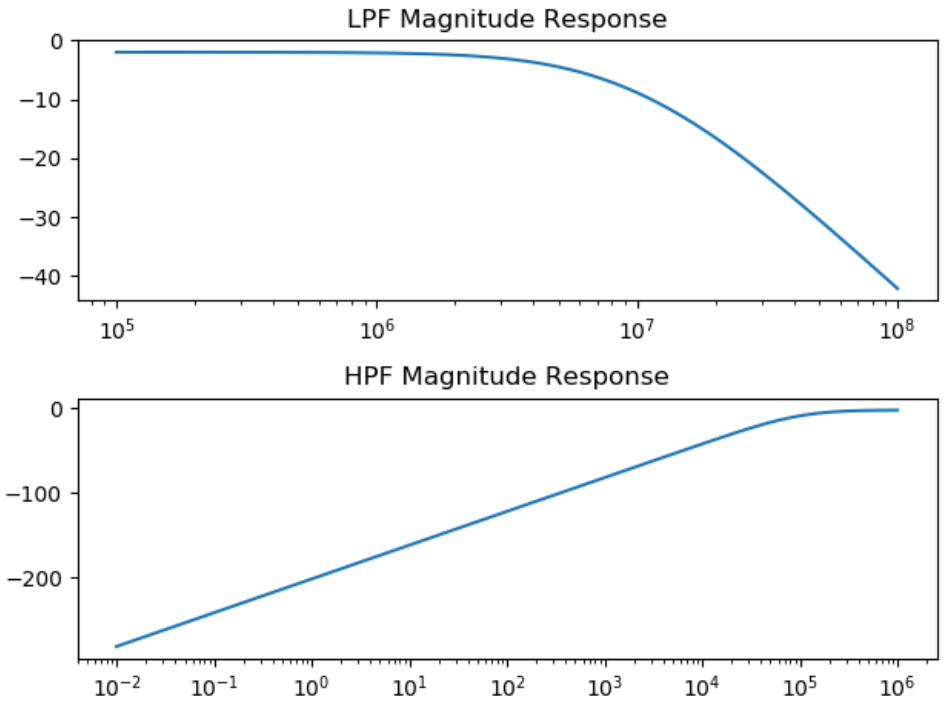
Format for running file on Terminal:

```
python3 EE18B104_Assign7_code.py
```

---

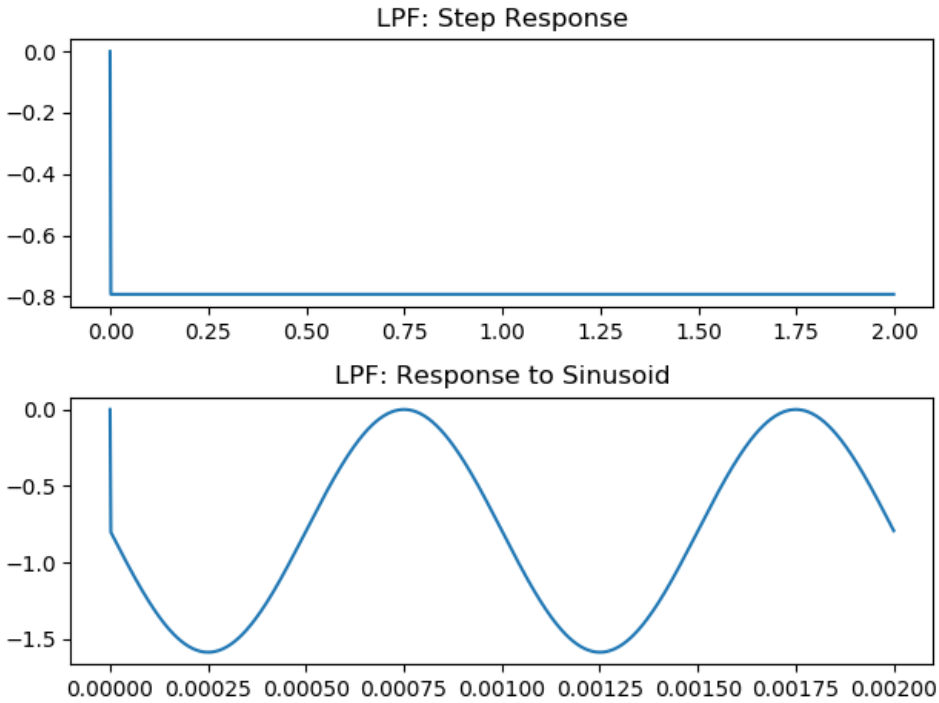
3 Plots

3.1 Bode Magnitude Plot of the Transfer Functions of the Filters

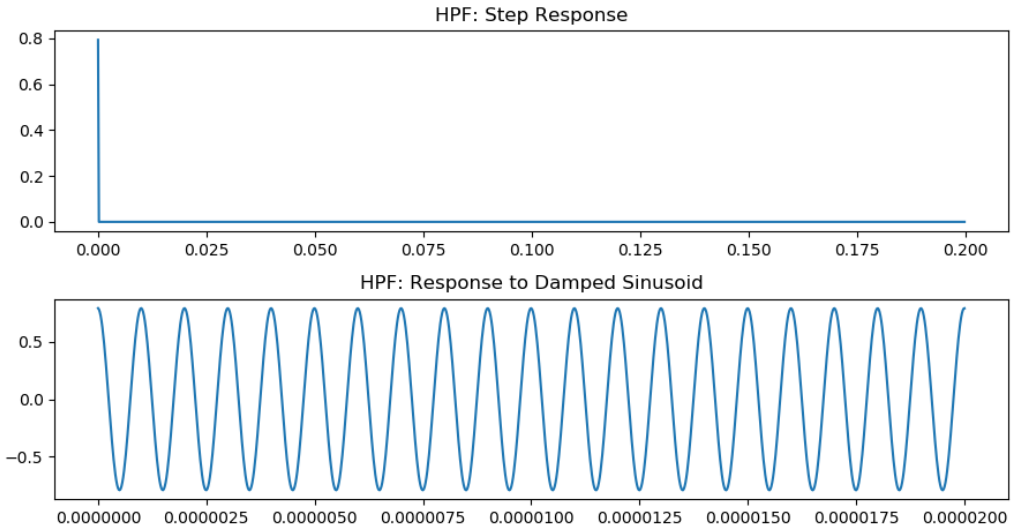


Parameters of the Low Pass Filter:
R1 = R2 = 10$k\Omega$, C1 = C2 = 10pF, K = 1.586 = open loop gain of Op-Amp
Parameters of the High Pass Filter:
R1 = R2 = 10$k\Omega$, C1 = C2 = 1nF, K = 1.586 = open loop gain of Op-Amp

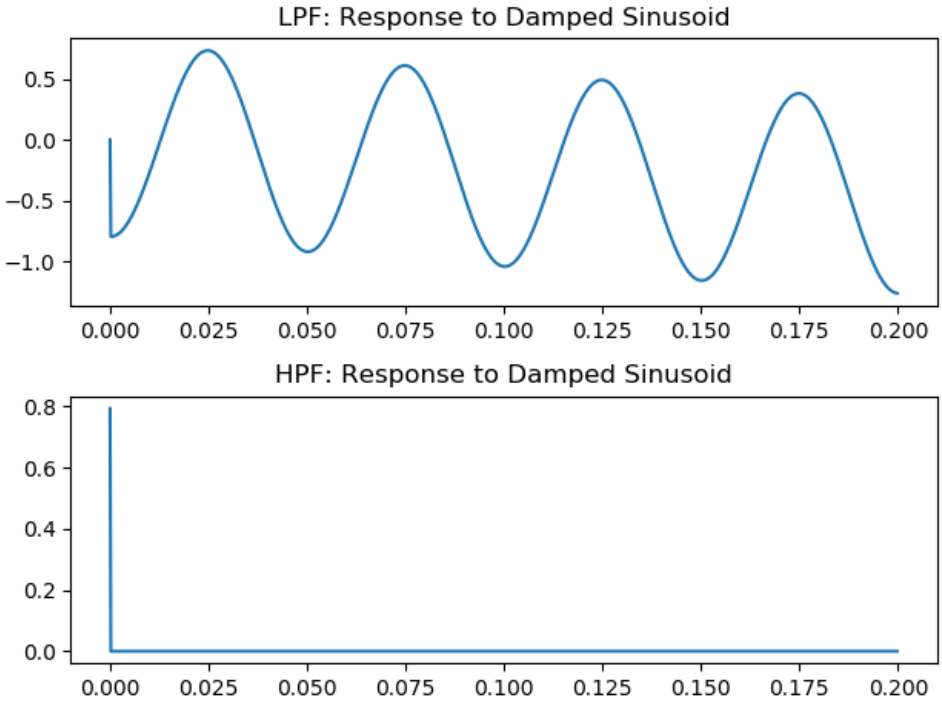
3.2 Response to Input signals



Input Sinusoid: $x(t) = (sin(2000\pi t) + cos(2 * 10^6 \pi t))u(t)$



Damped Sinusoidal Input Signal: $x(t) = cos(2 * 10^6 \pi t).exp(-t)u(t)$



Input Signal: $x(t) = (cos(40\pi t) + sin(\pi t).exp(-0.01t))u(t)$

---

4 Conclusion

1. We observe the expected behaviour of these filters from the obtained plots.

2. Note that the response to the damped signals quickly decay to 0. Hence the time vector used in sp.lsim is much smaller in comparison to that used for step response.

3. The magnitude gain for the filters in their intended frequencies (low frequencies for LPF and high frequencies for HPF) of operation is less than 1. This is due to the value of G chosen. For higher values of G, a high gain would be obtained.

4. Since u(t) (Heaviside function) is a semi-infinite signal, a truncated version (with length equal to the length of the time vector) is used. This gives rise to inaccuracies in the input response.

5. Another approach to this would have been to use the Laplace Transform of the Heaviside function, compute the Laplace Transform of y(t) and find its inverse. However, the Sympy function that computes the laplace transform of a time-symbolic expression (*sympy.integrals.transforms.laplace_transform*) was significantly slower. Hence the former approach was used.

⋆ ⋆ ⋆