Assignment-8 : Analog Filters

Sai Gautham Ravipati - EE19B053

April 26, 2021

Abstract

SymPy module in Python provides convenient way of solving algebraic equations. The same is used to analyse analog filters both high pass and low pass, by solving the conventional nodal analysis equations in s-domain. Once the transfer function has been obtained, analysis is carried out using scipy.signal module. Effectiveness of the module can be verified from the later parts of the assignment.

1 Low Pass Filter

A low pass filter amplifies low frequencies, while it attenuates high frequencies. The system given in the assignment acts a as a low pass filter. The nodal analysis equation is given by :

$$\begin{pmatrix} 0 & 0 & 1 & -1/G \\ \frac{-1}{sR_2C_2} & 1 & 0 & 0 \\ 0 & -G & G & 1 \\ \frac{-1}{R_1} - \frac{1}{R_2} - sC_1 & \frac{1}{R_2} & 0 & sC_1 \end{pmatrix} \begin{pmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{-V_i(s)}{R_1} \end{pmatrix}$$

This particular matrix equation can be solved in python using symbolic variables, and matrix algebra. The function call which implements the same is :

```
def lp_tf(R1,R2,C1,C2,G,Vi):
    """
    Solving matrix form of nodal analysis equation using sympy and the transfer
    function of
    low pass filter is returned.
    Inputs - Resistances  -  R1,R2
             Capacitances -  C1,C2
             Op-amp Gain  -  G
             Imput        -  Vi
    Output - Transfer function when Vi is 1
    """
    s = sy.symbols('s')
    A = sy.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 = sy.Matrix([0,0,0,-Vi/R1])
    V = A.inv()*b
    return V[3]
```

Since Vo is the fourth element of the matrix b, V[3] gives the transfer function when the input is 1, and the same has been returned by the function. The above given transfer function involves symbolic variables, to convert the same into coefficient vectors of numerator and denominator, so that they can be fed in sp.lti, which helps to analyse the system using signals toolbox, below given code does the work. The Hl and Hd sre the numerator and denominator coefficients of a polynomial transfer function. The magnitude and phase response of the filter are given in Fig 1,2.

```
def sy_to_sp(H_sy, s = sy.symbols('s')):
    """
    Converts a polynomial transfer function in sympy to numerator and denominator
    which can
    be fed into sp.lti.
    """
    n, d = sy.simplify(H_sy).as_numer_denom()              #Expressions of num and den
    p_nd = sy.Poly(n, s), sy.Poly(d, s)                    #Converting to polynomials
    c_nd = [p.all_coeffs() for p in p_nd]                  #Reading coefficients
    Hl, Hd = [sy.lambdify((),c)() for c in c_nd]           #Changing to floats
    return Hl,Hd
```

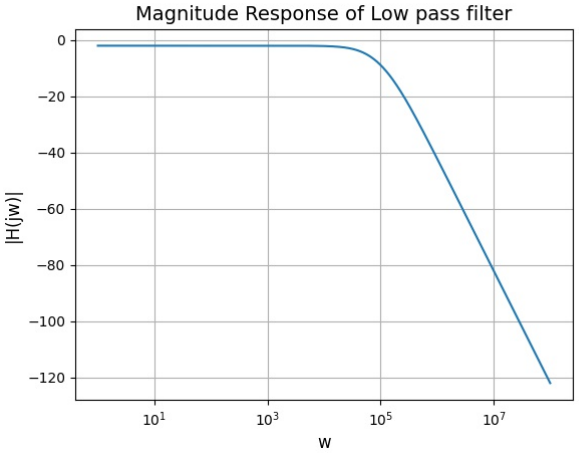

Figure 1: Magnitude response of LPF

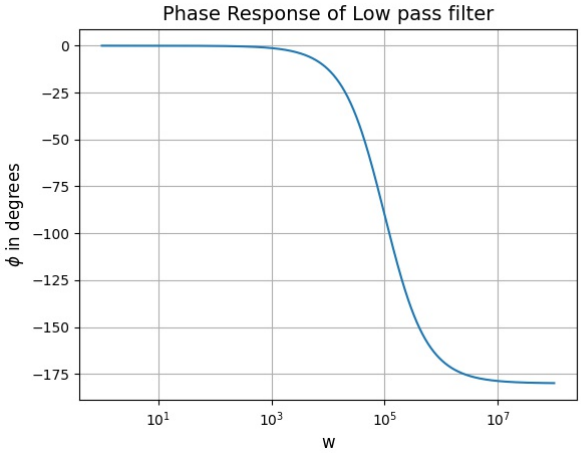

Figure 2: Phase response of LPF

Once we have the polynomial coefficients the following implementation is used obtain the step response.

```
def step_response(n,d,t):
    """
    Given the coefficient vector of numerator, denominator of the transfer function
    and time
    range, returns the step response in time domain.
    """
    H_sp = sp.lti(n,d)                                     #Transfer function
    F_sp = sp.lti(n, np.polymul(d,[1,0]))                  #Step Response in laplace domain
    _,x  = sp.impulse(F_sp,T = t)                          #Step response in time domain
    return x
```

The plot of step response obtained for $0 < t < 1ms$ is presented in Fig. 3. The step response is very much similar to that of the input, but saturates at a value close to 0.8 V, and this is very much same as the value of H(s) evaluated at s = 0 (can be verified by printing out the transfer function). However since input is changed from zero to one in this case, since output cannot change instantaneously because of the capacitve elements involved, there is some finite settling time associated with the output. Alternatively this can be modelled as DC component of the input along with some other frequency components being allowed by the filter, which gives the response which ultimately settles to a constant value.

The response to the sum of sinusoids can be obtained using sp.lism, and the output, for $0 < t < 10ms$, has been presented in Fig. 4. Clearly the output varies for 10 cycles in 10 ms, which means that the period is nearly 1 ms. This is because the LPF blocks the higher frequency while allows the lower frequency sinusoids. From the bode magnitude plot, gain is nearly 0dB and it starts to fall of at -40dB/decade around 10^5, thus since 2 kHz is less than 10^5, it is allowed while other frequency is

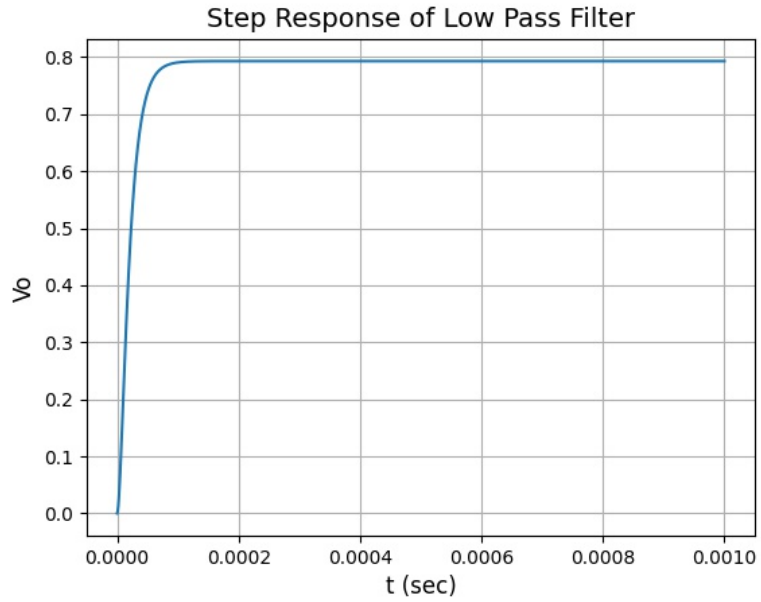

Figure 3: Step response of Low Pass Filter

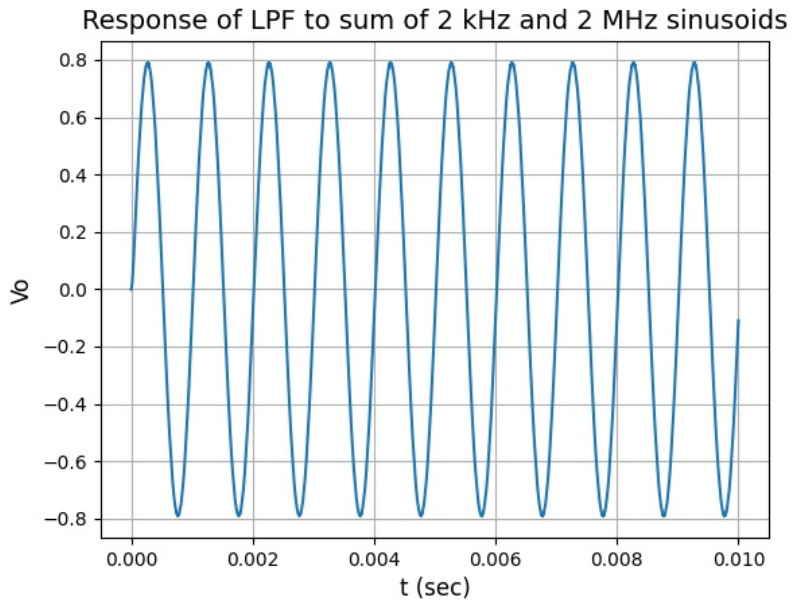

Figure 4: Response to sum of sinusoids of f = 2kHz and f = 2Mhz

blocked and the output is similar to that of only lower frequency sinusoid applied.

2 High Pass Filter

A high pass filter works exactly the opposite way in which the low pass filter works. It attenuates the low frequency components while it amplifies the high frequency components. The nodal analysis equation for a high pass filter is given as :

$$\begin{pmatrix} 0 & -1 & 0 & 1/G \\ \frac{sC_2R_2}{1+sC_2R_2} & 0 & -1 & 0 \\ 0 & G & -G & 1 \\ -sC_2 - \frac{1}{R_1} - sC_1 & 0 & sC_2 & \frac{1}{R_1} \end{pmatrix} \begin{pmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -V_i(s)sC_1 \end{pmatrix}$$

The same can be implemented using symbolic math and the solved using matrix algebra. The function call which implements the same is :

```
def hp_tf(R1,R2,C1,C2,G,Vi):
    """
    Solving matrix form of nodal analysis equation using sympy and the transfer function of
    high pass filter is returned.
    Inputs - Resistances  -  R1,R2 (R2 is R3 given in Circuit)
             Capacitances -  C1,C2
             Op-amp Gain  -  G
             Imput        -  Vi
    Output - Transfer function when Vi is 1
    """
    s = sy.symbols('s')
    A = sy.Matrix([[0,0,1,-1/G],[-s*R2*C2/(1+ s*R2*C2),1,0,0],[0,-G,G,1],[-1/R1-s*C1-s*C2,s*C2,0,1/R1]])
    b = sy.Matrix([0,0,0,-Vi*s*C1])
    V = A.inv()*b
    return V[3]
```

When input is 1, the value returned by the function is nothing but the transfer function of the high pass filter. The same can be converted to sp.lti transfer function using the function mentioned previously and .bode() is used to obtain the magnitude and phase data points, and the same has been plotted in a semi-log scale. The plots obtained are presented in Fig. 5,6. Clearly as inferred from the plots initially the magnitude plot raises as 40 dB/decade and begins to saturate ones it crosses the frequency 10^5. Thus this particular type of filter acts opposite to the way in which LPF acted previously, i.e. this filter block the low frequency components while amplifies the high frequency components. The same can be verified by passing the same input as of that in Q2. The response to sum of sinusoids is given in Fig. 7. Clearly the time period of the signal is close to $1\mu s$, indicating that lower frequency is attenuated.

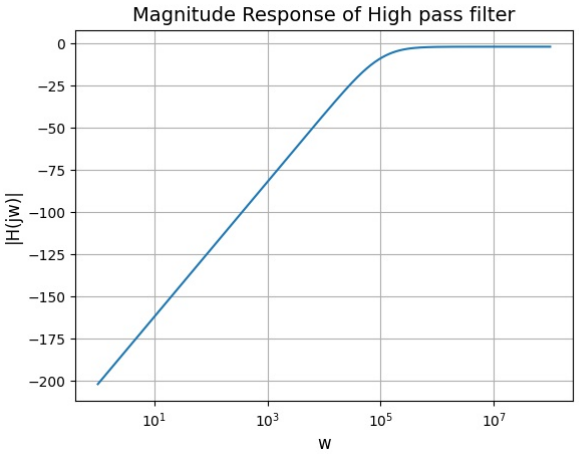

Figure 5: Magnitude response of HPF

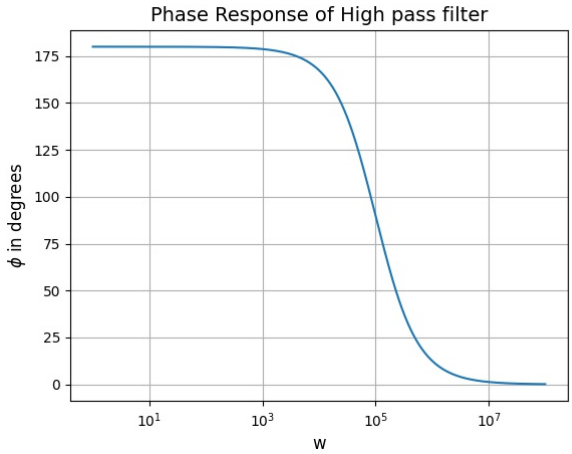

Figure 6: Phase response of HPF

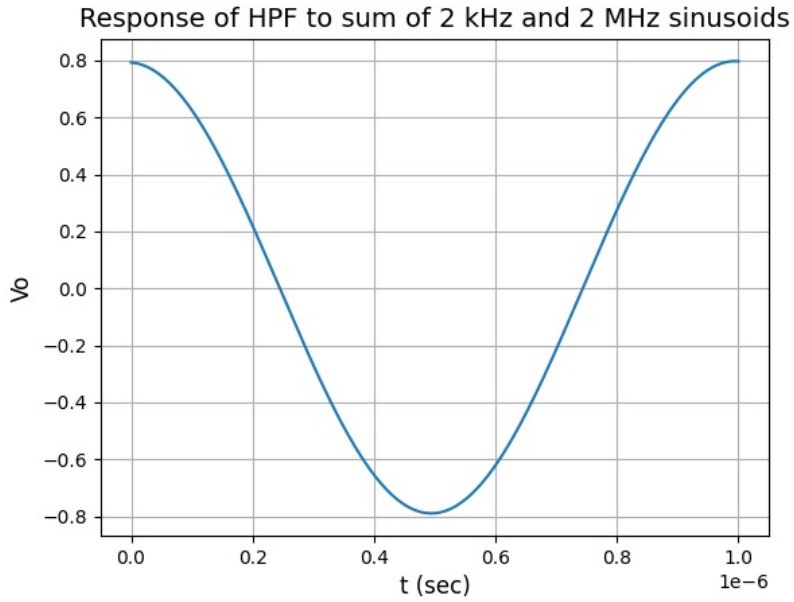

Figure 7: Response of HPF to sum of sinusoids

Consider the damped sinusoids of with different frequencies as below :

- $cos(2\pi * 10^7 t)e^{-1000t}u(t)$
- $cos(2\pi * 10^3 t)e^{-0.1t}u(t)$

The response of these signals is obtained using sp.lism, where impulse response is obtained to sp.lti as mentioned before. The same has been shown in Fig. 8,9,10.

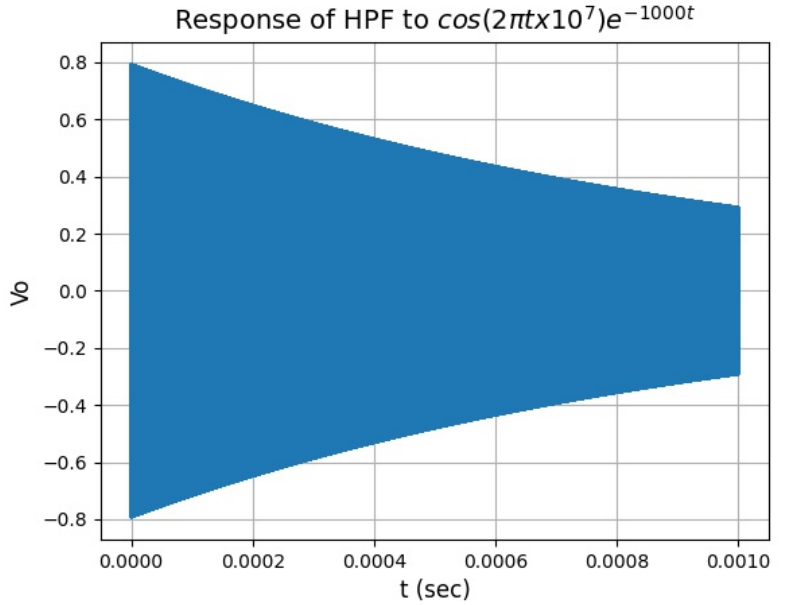

Figure 8: Response of HPF to 10MHz damped sinusoid

As inferred from the plot, there is almost no change when a high frequency damped sinusoid is passed through the HPF while its amplitude being changed by DC gain of filter. While lower

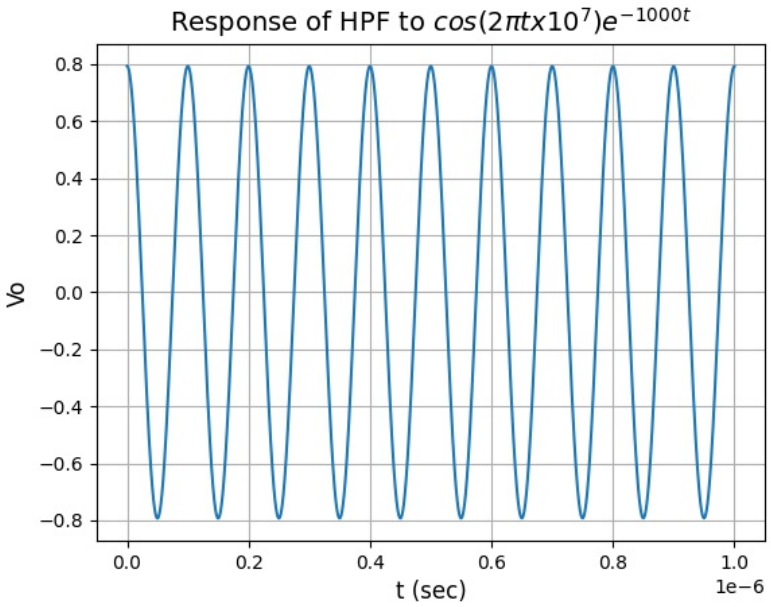

Figure 9: Response of HPF to 10MHz damped sinusoid (smaller interval)

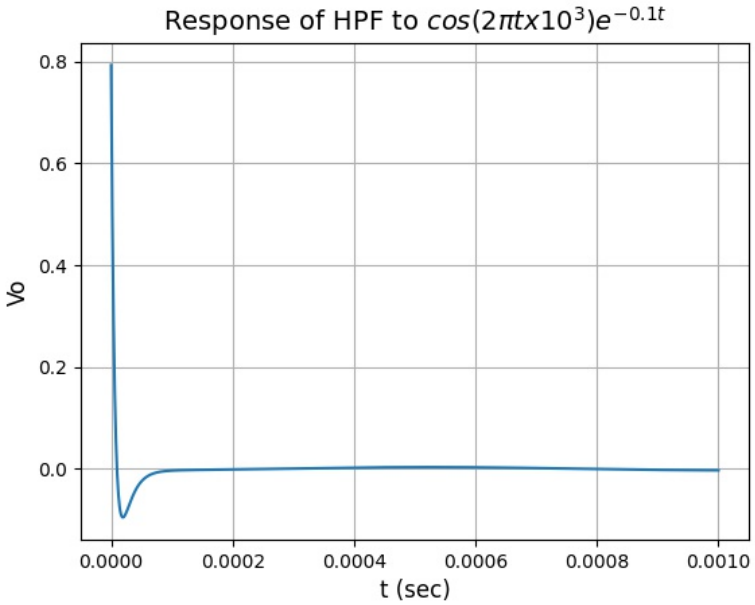

Figure 10: Response to HPF to 1KHz damped sinusoid

frequency damped sinusoid is attenuated, as the frequency is less than the knee frequency of the transfer function, however even this is not decayed to zero instantaneously, but involves some finite time to settle at 0.

The step-response of the high pass filter is obtained in a way similar to that of LPF, except that HPF transfer function is used in this case. The step-response has been presented in Fig 11. Initially when the signal is change from 0 to 1, this involves a non-DC component, as a result of which, the output of the step-response shows oscillation in the initial region, but once the signal settles at 1 V, it is similar to a constant DC being applied and since HPF attenuates at low frequencies the response approaches zero. Since both the 1KHz decaying sinusoid and unit-step are attenuated, both have

similar responses.

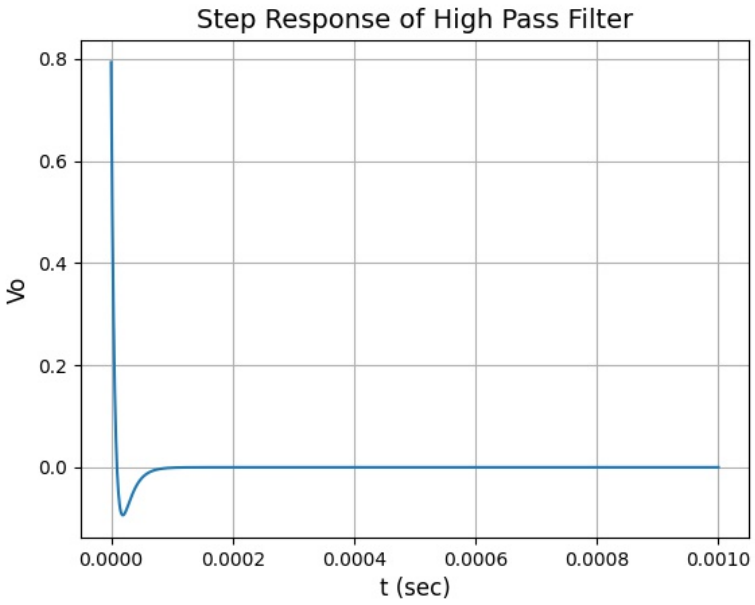

Figure 11: Step Response to HPF

3 Conclusion

Thus, SymPy provides an effective way of analysing circuits. Even though SymPy solves the matrix algebra, which integrate with scipy.signal acts as a powerful tool for circuit analysis. The low pass and high pass systems act opposite to each other as once block low frequencies other passes them. Several key-insights were drawn about the properties of Analog filters, and there step, magnitude, phase, response to sinusoids and damped sinusoids. Clearly, the low pass filter blocks high frequencies and high pass the low frequencies, and both of them have a finite transition band.