Assignment 7

Pruthvi Raj R G,EE17B114

March 28, 2019

In this week's assignment, the functions in sympy were used to solve the equations for a low-pass and a high-pass filter, employed using opamps, and as such get the transfer function of the system. Then, this transfer function was used to end the output waveform when an input with different frequency components were given.Using the functions that were used in last week's assignment, particularly scipy.signal.lsim and scipy.signal.impulse . The latter was used to end the waveform when the laplace transform of the output was known and the former was used to end the output when the input signal in the time domain is known, as well as the transfer function (in the frequency domain). Then, the input to the system was varied from a step function, to an exponentially decaying sinusoid and the various outputs that were obtained were noted.

1 Assignment Questions

1.1 Libraries and Global Variables Used

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

init_session
s = symbols('s')
```

1.2 Low-Pass Filter

In this, a low-pass filter circuit was realised using capacitors, resistors, and an opamp in negative feedback. The various KCL equations at the different nodes, that were used for solving the circuit are:

$$V_m = \frac{V_0}{G}$$

$$V_p = V_1 \frac{1}{1+jwR_2C_2}$$

$$V_0 = G(V_p - V_m)$$

$$\frac{V_i - V_1}{R_1} + \frac{V_p - V_1}{R_2} + jwC_1(V_0 - V_1) = 0$$

In the above equations, G is the ratio in which the output voltage is divided in the resistive feedback path of the opamp, as well as numerically equal to the gain of the opamp. Vp is the voltage at the positive terminal of the opamp, while Vm is that in the negative terminal. Vi is the input node of the system, and the resistor R1 is connected between this and node 1 (corresponding to V1). R2 is connected between V1 and Vp , while C2 is connected between Vp and ground. Finally, the capacitor C1 is connected between V 1 and V o , in another feedback path that would be blocked for DC signals. These above equations were solved using matrices in sympy using the following code:

The code is as follows:

```
def lowpass(R1,R2,C1,C2,G,Vi):
    s=symbols('s')
    A=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=Matrix([0,0,0,-Vi/R1])
    V=A.inv()*b
    return (A,b,V)
```

In this, matrix A is the coeffcient matrix and B is the matrix having the independent sources.The various functions related to calculating the time domain output are declared below. First, is the one that returns the frequency domain output of the system for a particular input. It takes 2 parameters: the laplace transform of the input and filterType, which is whether to use highpass or lowpass filter. High pass filter will be implemented in the next section.

```
def highpass(R1,R3,C1,C2,G,Vi):
    s=symbols('s')
    A=Matrix([[0,0,1,-1/G],[-(s*R3*C2)/(1+s*R3*C2),1,0,0],
    [0,-G,G,1],[-(s*C1)-(s*C2)-(1/R1),s*C2,0,1/R1]])
    b=Matrix([0,0,0,-Vi*s*C1])
    V=A.inv()*b
    return (A,b,V)
```

```
def get_response_freq_domain(h,filterType):
    if (filterType == 'l'):
        A,b,V = lowpass(10000,10000,1e-9,1e-9,1.586,h)
    elif (filterType == 'h'):
        A,b,V = highpass(10000,10000,1e-9,1e-9,1.586,h)
    Vo=V[3]


    w=np.logspace(0,8,801)
    ss=1j*w
    hf=lambdify(s,Vo,'numpy')
    v=hf(ss)
    return Vo,w,abs(v),np.angle(v)



def get_numpy_array_from_Poly(num,den):
    isFloat = False
    try:
        num = Poly(num).all_coeffs()
    except GeneratorsNeeded:
        num = num
        isFloat = True
    den = Poly(den).all_coeffs()


    den2 = []
    num2 = []
    for i in den:
        den2.append(float(i))
    den2 = np.array(den2)

    if (isFloat):
        num2 = num
    else:
        for i in num:
            num2.append(float(i))
        num2 = np.array(num2)
    return num2,den2


def get_output_time_domain(Y,t,steps):
    simplY = simplify(Y)
    num = fraction(simplY)[0]
```

```
    den = fraction(simplY)[1]
    num2,den2 = get_numpy_array_from_Poly(num,den)

    num2 = np.poly1d(num2)
    den2 = np.poly1d(den2)


    Y = sp.lti(num2,den2)
    t = np.linspace(0.0,t,steps)
    t,y=sp.impulse(Y,None,t)
    return t,y
```

The second function is called from the main program for calculating the output, which in turn calls the first function to end and return the coeffcients of the laplace transform of the output, so that scipy.signal.impulse can calculate the output correctly. The above function can be used if the laplace tranform of the input is known. If, only the time domain representation of the input is known, the following function can be used to calculate the output using scipy.signal.lsim instead of scipy.signal.impulse . The function takes 3 parameters: The sympy expression of the transfer function, time domain input wave as well as the time interval to calculate output.

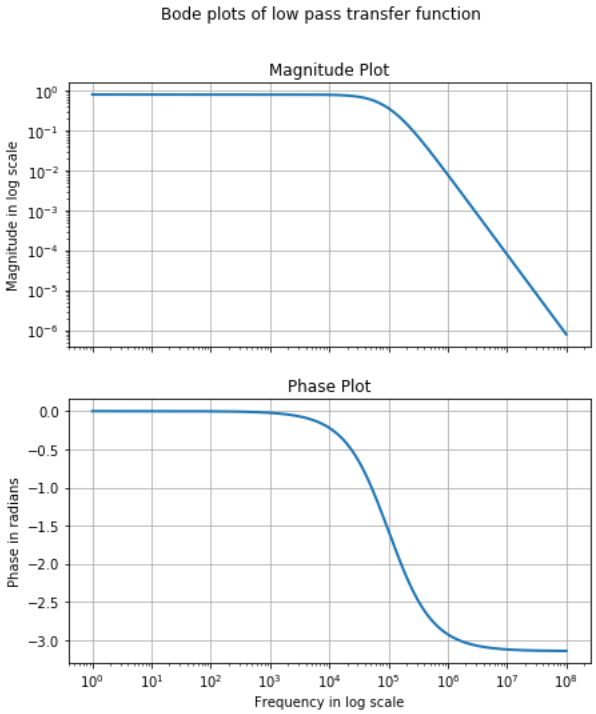
```
def get_output_with_lsim(H,x,t):


    simplH = simplify(H)
    num = fraction(simplH)[0]
    den = fraction(simplH)[1]
    num2,den2 = get_numpy_array_from_Poly(num,den)

    num2 = np.poly1d(num2)
    den2 = np.poly1d(den2)

    H = sp.lti(num2,den2)
    t,y,sec=sp.lsim(H,x,t)
    return t,y
```

1.3 Transfer function of system

If, an input Vi is given as an impulse (i.e laplace transform input is 1), then, the transfer function of the above system can be known. Thus, the transfer function, so calculated and plotted is:.

```
Vi = 1
```

```
H_l,w,v,ph = get_response_freq_domain(Vi,'l')


fig, axes = plt.subplots(2, 1, figsize=(7, 8), sharex = True)
plt.suptitle('Bode plots of low pass transfer function')
axes[0].loglog(w,v,lw=2)
axes[0].grid()
axes[0].set_ylabel('Magnitude in log scale')
axes[0].set_title('Magnitude Plot')

axes[1].semilogx(w,ph,lw=2)
axes[1].grid()
axes[1].set_xlabel('Frequency in log scale')
axes[1].set_ylabel('Phase in radians')
axes[1].set_title('Phase Plot')
plt.show()
```

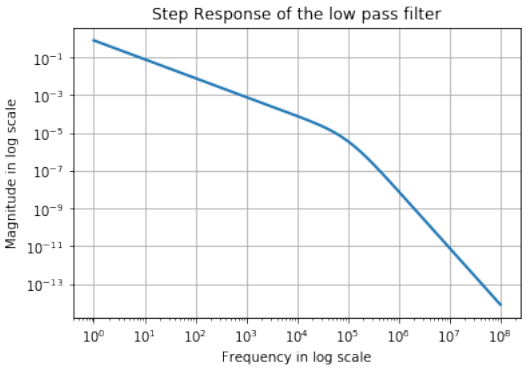


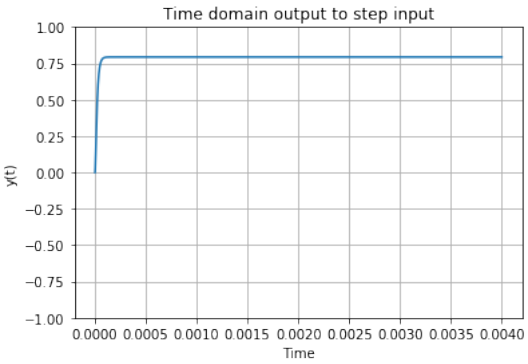
Thus, it is evident that, the system above acts as a low pass filter, as it passes low frequencies with almost no attenuation, while highly attenuates high frequency inputs. Also, the 3dB bandwidth (magnitude at phase of pi/2), is around 100000 Hz, which means that, any frequency below this, it will pass with minimal attenuation.

1.4 Step Response

The response of the low pass filter to a step input is calculated. For this, the laplace transform of the step input is used as input to the above functions and the step response in both time and frequency domains are plotted. Code for finding the above:

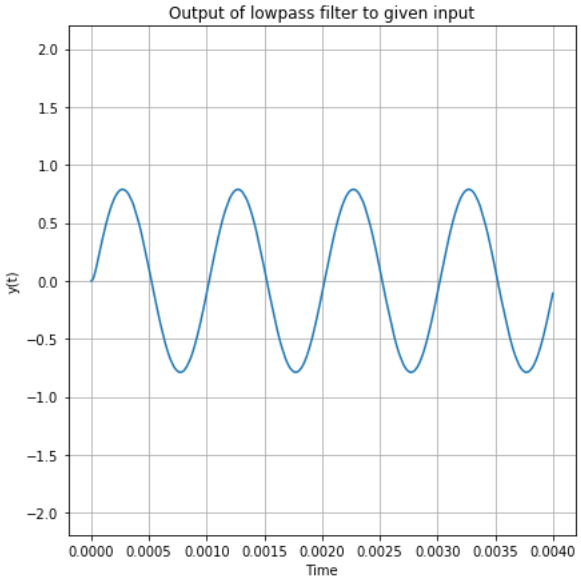
```
Vi = 1/s
Y,w,v,ph = get_response_freq_domain(Vi,'l')
t,y = get_output_time_domain(Y,4e-3,10001)


plt.loglog(w,v,lw=2)
plt.grid()
plt.title('Step Response of the low pass filter')
plt.xlabel('Frequency in log scale')
plt.ylabel('Magnitude in log scale')
plt.show()


plt.plot(t,y)
plt.grid()
plt.title('Time domain output to step input')
plt.xlabel('Time')
plt.ylabel('y(t)')
plt.ylim(-1,1)
plt.show()
```

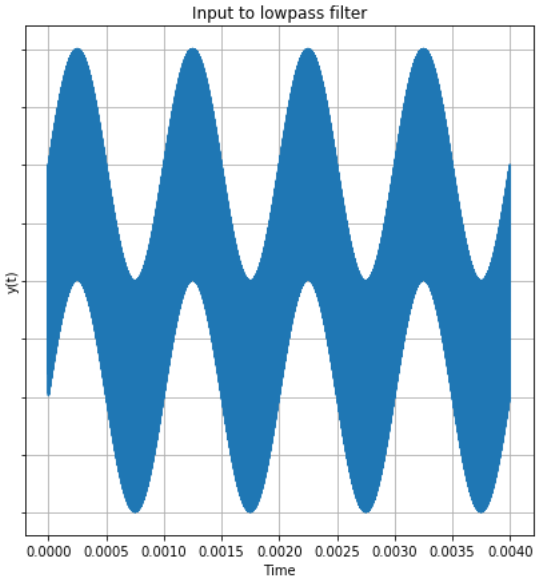




1.5 Given Input

For this, the lsim function is used, wherein we know the time domain input, and the transfer function of the system (calculated above as H). Using these and scipy.signal.lsim , the output time domain wave is calculated. In the below code segment, both the input and output waves are plotted for comparison






```
t = np.linspace(0.0,4e-3,100001)
x = np.sin(2000*math.pi*t) + np.cos(2*(10**6)*math.pi*t)
t,y = get_output_with_lsim(H_l,x,t)

fig, axes = plt.subplots(1, 2, figsize=(15, 7), sharey = True)
axes[0].plot(t,y)
```

```
axes[0].grid()
axes[0].set_title('Output of lowpass filter to given input')
axes[0].set_xlabel('Time')
axes[0].set_ylabel('y(t)')

axes[1].plot(t,x)
axes[1].grid()
axes[1].set_title('Input to lowpass filter')
axes[1].set_xlabel('Time')
axes[1].set_ylabel('y(t)')
plt.show()
```

Thus, it is found that, although input had high frequency components, the circuit filters all of this, and gives output as only the low frequency component.

1.6 High-Pass Filter

The only difference in this implementation is that the KCL equations used for calculating V o would be different. The KCL equations for a high pass filter using opamps, resistors and capacitors are

$$V_m = \frac{V_0}{G}$$

$$V_p = V_1 \frac{jwR_3C_2}{1+jwR_3C_2}$$

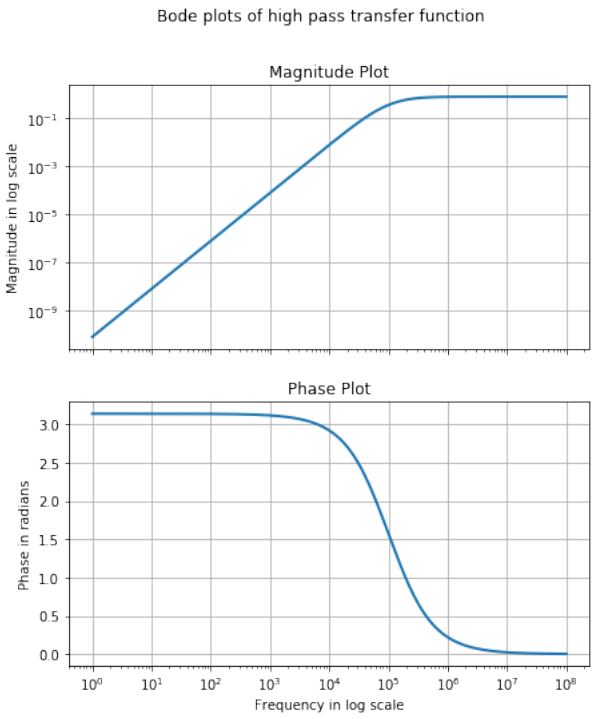
$$V_0 = G(V_p - V_m)$$

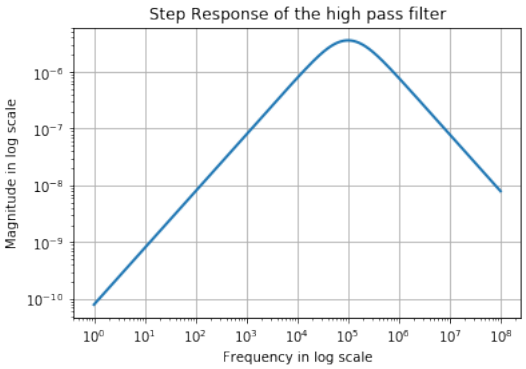
$$jwC_1(V_i - V_1) + jwC_2(Vp - V_1) + \frac{V_0 - V_1}{R_1} = 0$$

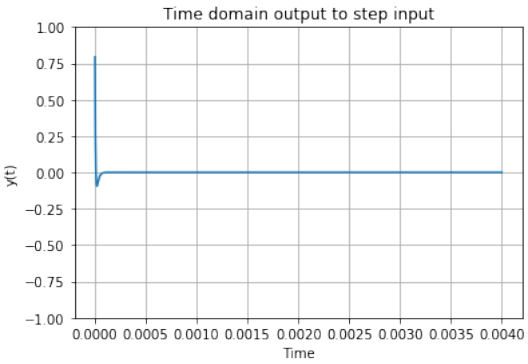
In the above equations, G is the ratio in which the output voltage is divided in the resistive dividing feedback path of the opamp, as well as numerically equal to the gain of the opamp. Vp is the voltage at the positive terminal of the opamp, while Vm is that in the negative terminal. V i is the input node of the system, and the capacitor C1 is connected between this and node 1 (corresponding to V1). C2 is connected between V1 and Vp , while R3 is connected between Vp and ground. Finally, the resistor R1 is connected between V 1 and V o , in another feedback path. A python function is declared that solves the above equations similar to that done for lowpass filter.

```
def highpass(R1,R3,C1,C2,G,Vi):
    s=symbols('s')
    A=Matrix([[0,0,1,-1/G],[-(s*R3*C2)/(1+s*R3*C2),1,0,0],
```

```
[0,-G,G,1],[-(s*C1)-(s*C2)-(1/R1),s*C2,0,1/R1]])
b=Matrix([0,0,0,-Vi*s*C1])
V=A.inv()*b
return (A,b,V)
```



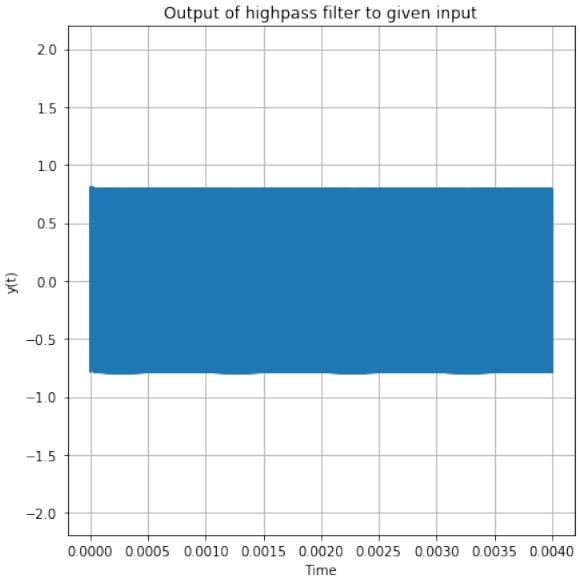


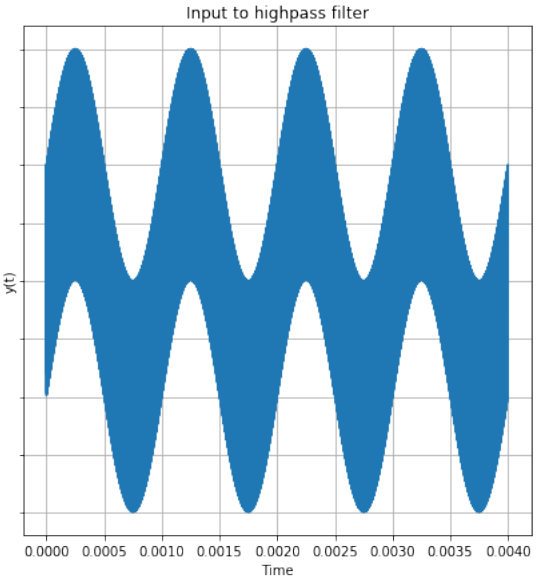


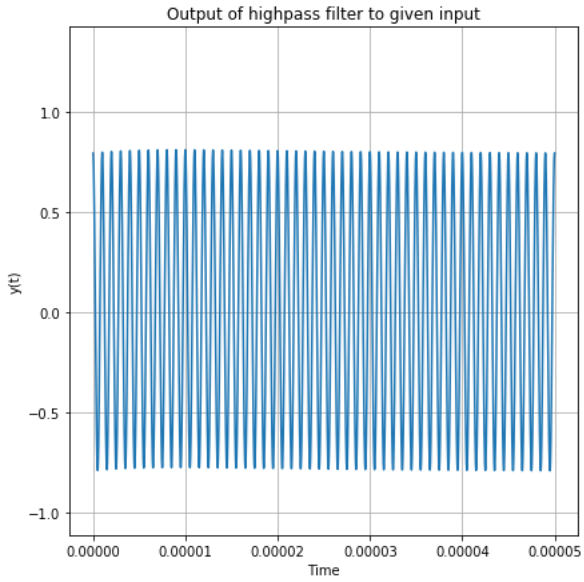
All the other functions that were declared to calculate and plot the output transforms and signals can be used here also, as except the equations, nothing else changes in the overall logic of the program.

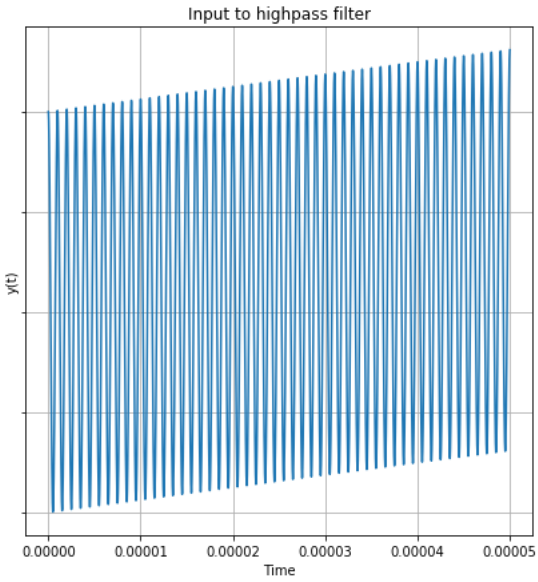
1.7 Input as(sin(2000t) + cos(210e6t))u0(t)

In this, the same input, which was given to the lowpass filter, is given to the highpass filter. In the lowpass filter case, it was seen that, only the low frequency component was passed, so here the expected output is that only the high frequency component should pass.



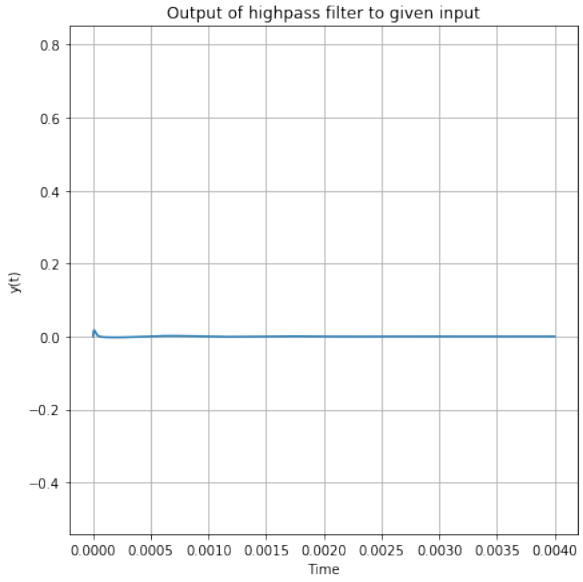


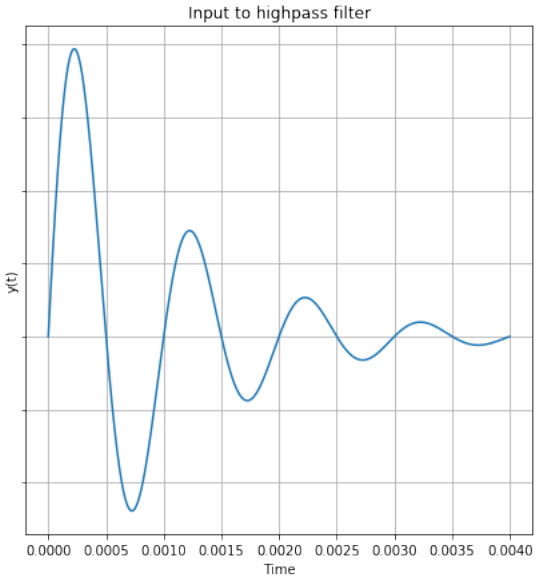




1.8 Low frequency decay wave

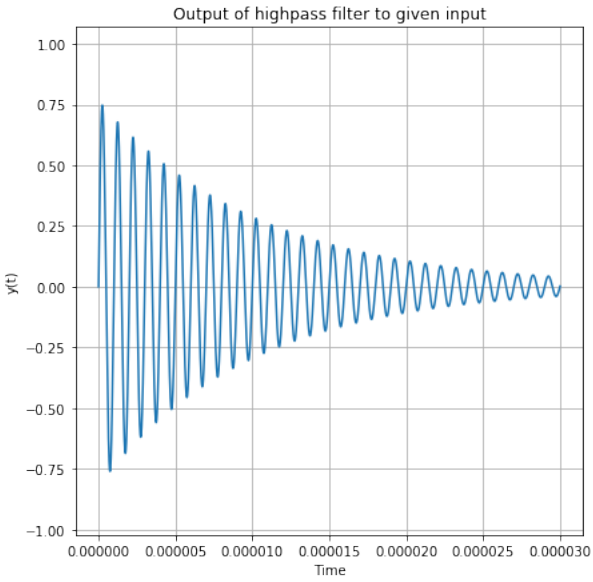
A decaying wave of frequency 2000 Hz, is given as input to the high pass filter and output obtained is observed. The input given is sin (2000t)e 1000t

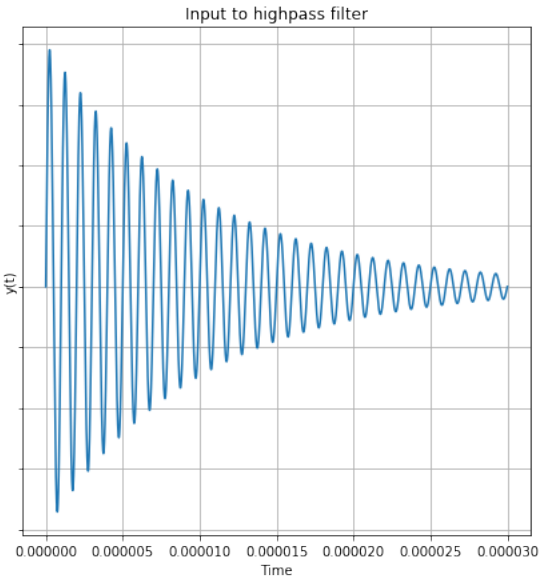




1.9 High frequency deacy wave

Here, a high frequency wave of frequency 2M is given.





1.10 Conclusion

For the other signal input to the filter, it is seen that, the low pass filter passes only the low frequency component and almost completely attenuates the high frequency part, as expected. Also, from the Bode plots of the transfer function of the low pass filter, it is observed that as the low frequency component of the input (1000 Hz) is less than the 3dB bandwidth of the system it passes with little attenuation (a gain of around 0.8), whereas for that of the high frequency component (10e6 Hz), which is higher than the 3dB bandwidth of around 10e5 , gain is almost 10e2 , and hence is not visible in the graph

From the step-response, of the low pass filter, it is seen that, at steady state, output is a constant value (at a very low attenuation). This is because, low pass filter will pass DC signals , and at time t much greater 0, step input is a DC input.At t = 0, though the rise is gradual, as the capacitors themselves take time to charge, and once they have done, the output becomes DC.

From the step-response of the high pass filter, it is seen that at t much greater than 0, output is 0. This is because, at these instants, input is a DC value and the high pass filter will attenuate this, and thus output will be almost 0. At t = 0, though, there is a peak for the output. This is because, at this point of discontinuity in the input, as seen from the frequency domain, a lot of high frequency components would be there, hence these would be passed, and so, an output is observed only at this point. These high frequency components are due to the discontinuity, which cannot be

replicated using the fourier transform as the fourier transform decomposes the function into a bunch of continuous time sinusoids, hence, Gibb's phenomenon occurs at the discontinuity, causing the initial peak in the step response.