EE2703 Applied Programming Lab - Assignment 8

Name: Rajat Vadiraj Dwaraknath
Roll Number: EE16B033

April 1, 2018

1 Introduction

In this assignment, we use Sympy to analytically solve a matrix equation governing an analog circuit. We look at two circuits, an active low pass filter and an active high pass filter. We create matrices using node equations for the circuits in sympy, and then solve the equations analytically. We then convert the resulting sympy solution into a numpy function which can be called. We then use the signals toolbox we studied in the last assignment to understand the responses of the two circuits to various inputs.

2 Low pass filter circuit

We create a function to solve the low pass active filter circuit given in the assignment question as the first figure.

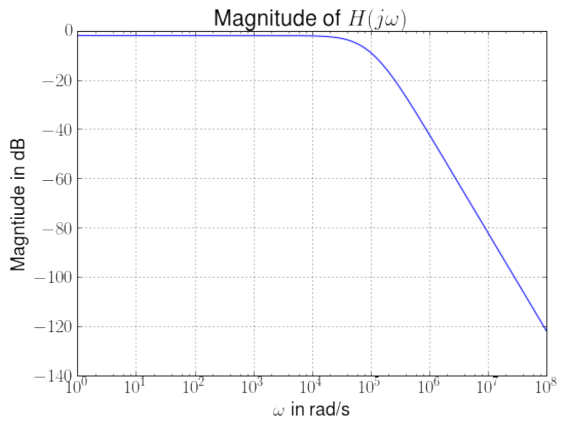
```
from sympy import *
import scipy.signal as sp

H,s=symbols('H(s) s')
init_printing()

def lowpass(R1=10e3,R2=10e3,C1=1e-9,C2=1e-9,G=1.586,Vi=1):
    """Solve the given lowpass filter circuit for a given input Vi."""
    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)
```

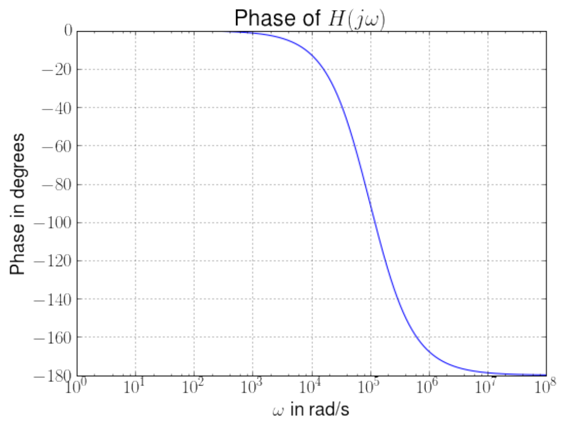
A helper function to graph the bode plots of an arbitrary sympy expression in s is written below:

```
def bodePlot(H_s,w_range=(0,8),points=800):
    """Plot the magnitude and phase of H_s over the given range of frequencies."""

    w = logspace(*w_range,points)
    h_s = lambdify(s,H_s,'numpy')
    H_jw = h_s(1j*w)

    # find mag and phase
    mag = 20*np.log10(np.abs(H_jw))
    phase = angle(H_jw,deg = True)

    eqn = Eq(H,simplify(H_s))
    display(eqn)

    fig,axes = plt.subplots(1,2,figsize=(18,6))
    ax1,ax2 = axes[0],axes[1]

    # mag plot
    ax1.set_xscale('log')
    ax1.set_ylabel('Magntiude in dB')
    ax1.set_xlabel('$\omega$ in rad/s')
    ax1.plot(w,mag)
    ax1.grid()
    ax1.set_title("Magnitude of $H(j \omega)$")

    # phase plot
    ax2.set_ylabel('Phase in degrees')
    ax2.set_xlabel('$\omega$ in rad/s')
    ax2.set_xscale('log')
    ax2.plot(w,phase)
    ax2.grid()
    ax2.set_title("Phase of $H(j \omega)$")

    plt.show()
```

The bode plots of the low pass filter transfer function, along with the analytical expression of the transfer function are shown below:

$$H(s) = \frac{0.0001586}{2.0 \cdot 10^{-14} s^2 + 4.414 \cdot 10^{-9} s + 0.0002}$$





We observe that the gain rolls off at -40 dB per decade after the cutoff frequency of around 10^5 rad/s. The phase also changes directly from an initial value of 0 degrees to a final value of -180 degrees. This means that the filter is a second order filter. Since the phase starts of at 0 degrees, this filter is a non-inverting low pass filter. Also note that the DC gain is approximately 0.793, and the AC gain is 0.

Let us analyze the system in terms of its quality factor.

A function to convert a rational polynomial sympy expression into a tuple of numerator and denominator coefficients is written below:

```python
def symToTransferFn(Y_s):
    """
    Convert a sympy rational polynomial into one that can be used for scipy.signal.lti.

    Returns a tuple (num,den) which contains the coefficients of s of
    the numerator and denominator polynomials.
    """

    # get the polynomial coefficients after simplification
    Y_ss = expand(simplify(Y_s))
    n,d = fraction(Y_ss)
    n,d = Poly(n,s), Poly(d,s)
    num,den = n.all_coeffs(), d.all_coeffs()
    num,den = [float(f) for f in num], [float(f) for f in den]

    return num,den
```

A function to find the Q factor of a second order system is written below:

```python
def findQ(H_s):
    """Find the quality factor of the input transfer function assuming it is second order."""
    nl,dl = symToTransferFn(H_s)
    syst = sp.lti(nl,dl)
    p1,p2 = syst.poles[0], syst.poles[1]
    return np.sqrt(abs(p1*p2))/abs(p1+p2)
```

```
print("Q factor of low pass filter: {:.4f}".format(findQ(lowpass()[-1][-1])))
```

```
Q factor of low pass filter: 0.4531
```

With a Q factor of approximately 0.45 which is quite close to half, we can conclude that the filter is almost critically damped. This means that it has a very fast response time as the transients in its output die off as fast as possible.

3 High pass filter circuit

We now repeat the process but for the high pass filter circuit given as the last figure in the assignment question pdf. Note that the signs of the gain block are incorrectly written in the question pdf, so they are switched in the subsequent analysis. The feedback path through the two resistors $(G-1)R$ and R connects to the negative terminal of the gain block. To solve the circuit, we combine the node equations at nodes V_1, V_p and V_n along with the output equation of the differential gain block into one matrix equation below:

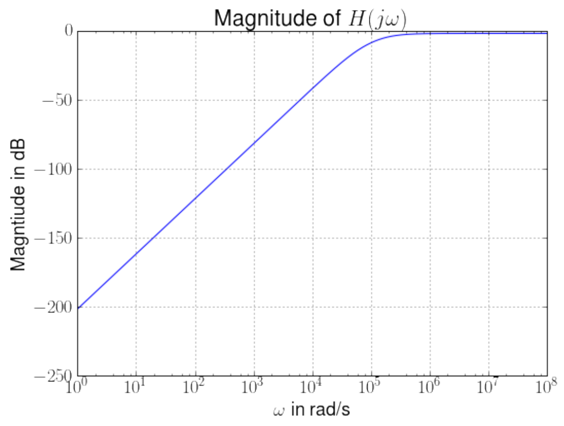
$$\begin{bmatrix} 0 & -1 & 0 & \frac{1}{G} \\ \frac{C_2R_3s}{C_2R_3s+1} & 0 & -1 & 0 \\ 0 & G & -G & 1 \\ -C_1s-C_2s-\frac{1}{R_1} & 0 & C_2s & \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_n \\ V_p \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -C_1V_is \end{bmatrix}$$

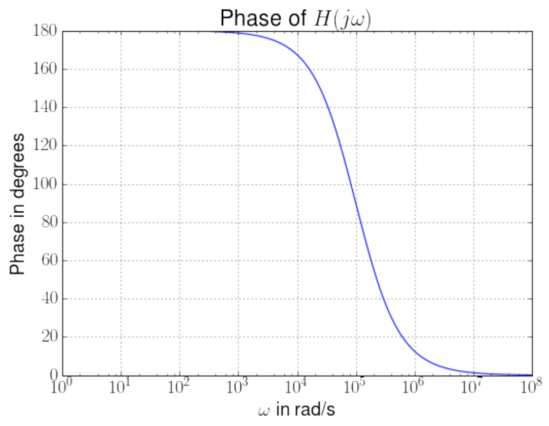
We now define a function to analytically solve this equation in sympy:

```
#G1 = symbols('G1')
def highpass(R1=10e3,R3=10e3,C1=1e-9,C2=1e-9,G=1.586,Vi=1):
    """Solve the given highpass filter circuit for a given input Vi."""

    A=Matrix([[0,-1,0,1/G],
        [s*C2*R3/(s*C2*R3+1),0,-1,0],
        [0,G,-G,1],
        [-s*C2-1/R1-s*C1,0,s*C2,1/R1]])

    b=Matrix([0,0,0,-Vi*s*C1])

    V=A.inv()*b
    #V_lim = [limit(v,G1,oo) for v in V]
    return (A,b,V)
```

The bode plots of the transfer function and its analytical expression are shown below:

$$H(s) = \frac{1.586 \cdot 10^{-14}s^2}{2.0 \cdot 10^{-14}s^2 + 4.414 \cdot 10^{-9}s + 0.0002}$$





We observe from the gain plot that the gain rolls off at +40 dB per decade before the cutoff frequency of around 10^5 rad/s. This means that it is second order high pass filter. Since the phase at very high frequencies is approximately 0 degrees, this circuit is a non-inverting high pass filter. Also note that the AC gain is approximately 0.793, and the DC gain is 0.

```
print("Q factor of high pass filter: {:.4f}".format(findQ(highpass()[-1][-1])))
```

```
Q factor of high pass filter: 0.4531
```

The Q factor of this filter is the same as that of the low pass filter. This means that this filter is also almost critically damped, which means it responds as fast as possible to sudden changes in inputs.

4 Responses to inputs

We find the responses of the above two filter circuits to various inputs by converting the sympy expressions to numpy functions using **lambdify**. We then use the signals toolbox to find the response in the time domain.

A function to find the causal inverse laplace transform of a sympy expression using **scipy.signal.impulse** is written below:

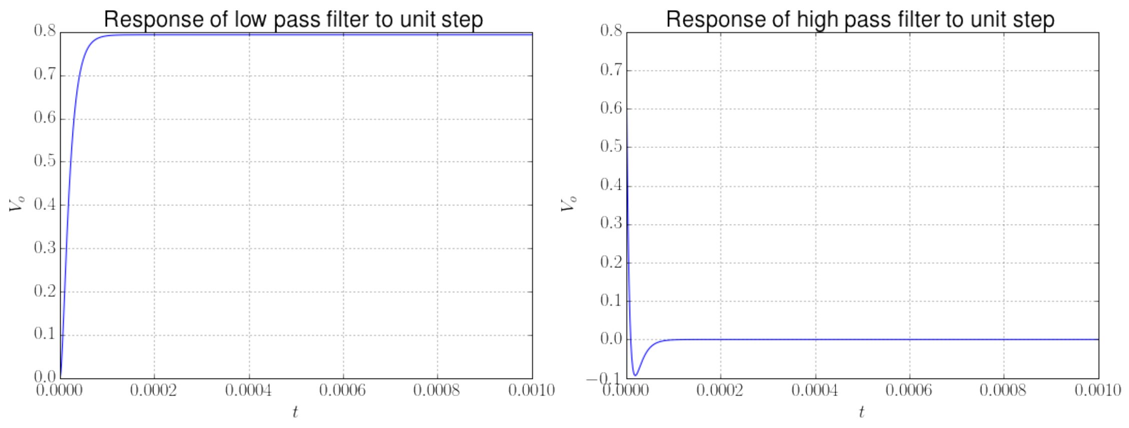
```
def inverseLaplace(Y_s,t=None):
    """
    Finds the inverse laplace transform of a sympy expression using sp.impulse.
    """

    # load the step response as a system in scipy.signal
    num,den = symToTransferFn(Y_s)

    # evaluate in time domain
    t,y = sp.impulse((num,den),T=t)
    return t,y
```

A function to plot the time domain responses of the two filter circuits to an arbitrary input specified in either the Laplace domain or the time domain is written below:

```
def plotFilterOutputs(laplace_in=None, time_domain_fn=None,
                      lp_range=(0,1e-3), hp_range=(0,1e-3), points=1e3,
                     input_name="Input"):
    """
    Plot the time domain outputs of the two active filters to a given input in the
    laplace domain or the time domain.
    """

    t_lp = linspace(*lp_range,points)
    t_hp = linspace(*hp_range,points)


    if laplace_in != None:

        A,b,V_lowpass = lowpass(Vi=laplace_in)
        t_lp,y_lp = inverseLaplace(V_lowpass[-1],t=t_lp)


        A,b,V_highpass = highpass(Vi=laplace_in)
        t_hp,y_hp = inverseLaplace(V_highpass[-1],t=t_hp)

    elif time_domain_fn != None:

        A,b,V_lowpass = lowpass()
        lowsys = symToTransferFn(V_lowpass[-1])
        t_lp,y_lp,svec = sp.lsim(lowsys, time_domain_fn(t_lp), t_lp)


        A,b,V_highpass = highpass()
        highsys = symToTransferFn(V_highpass[-1])
        t_hp,y_hp,svec = sp.lsim(highsys, time_domain_fn(t_hp), t_hp)

    else:
        print("No input given.")


    fig,axes = plt.subplots(1,2,figsize=(18,6))
    ax1,ax2 = axes[0],axes[1]

    # low pass response plot
    ax1.set_ylabel('$V_o$')
    ax1.set_xlabel('$t$')
    ax1.plot(t_lp,y_lp)
    ax1.grid()
    ax1.set_title("Response of low pass filter to {}".format(input_name))
    # high pass response plot
    ax2.set_ylabel('$V_o$')
    ax2.set_xlabel('$t$')
    ax2.plot(t_hp,y_hp)
    ax2.grid()
    ax2.set_title("Response of high pass filter to {}".format(input_name))

    plt.show()
    return t_lp,y_lp, t_hp,y_hp
```

4.1 Step response

We plot the outputs of the two systems to a unit step below:



```
Steady state value of low pass filter step response: 0.7930
Steady state value of high pass filter step response: 0.0000
Initial value of low pass filter step response: 0.0000
Initial value of high pass filter step response: 0.7930
```

- We observe that the low pass filter inverts the step and attenuates it by a factor of 0.793. This is indeed what we expect as we noticed earlier that the DC gain of the transfer function is 0.793. We observe a transient which decays quite fast, again as we expected as the system is almost critically damped.
- The initial value of the low pass response to the step is 0 as the AC gain of the low pass filter is 0. We can interpret the step as an infinte frequency signal, so the response to it would be according to the AC gain.
- The transient of the high pass response is also similar. There are two differences. Firstly, the steady state response of the high pass filter to the step is 0. This is because it only allows frequencies higher than the cutoff to pass through. Since DC inputs have a frequency of 0, it is completely filtered out. In other words, the DC gain is 0. This is similar to what is done when two systems are coupled for AC signals. The initial value of the high pass filter response is 0.793. This is because the AC gain of the filter is 0.793.

- The other difference is that the response overshoots the steady state value of 0, reaches an extremum, then settles back to 0, unlike the response of the low pass filter which steadily approaches the steady state value with no extrema. This occurs because of the presence of zeros at the origin in the transfer function of the high pass filter(which imply that the DC gain is 0). Since the steady state value of the step response is 0, the total signed area under the curve of the impulse response must also be 0. This means that the impulse response must equal zero at one or more time instants. Since the impulse response is the derivative of the step response, this therefore means that the step response must have at least one extremum. This explains the behaviour of the step response of the high pass filter.

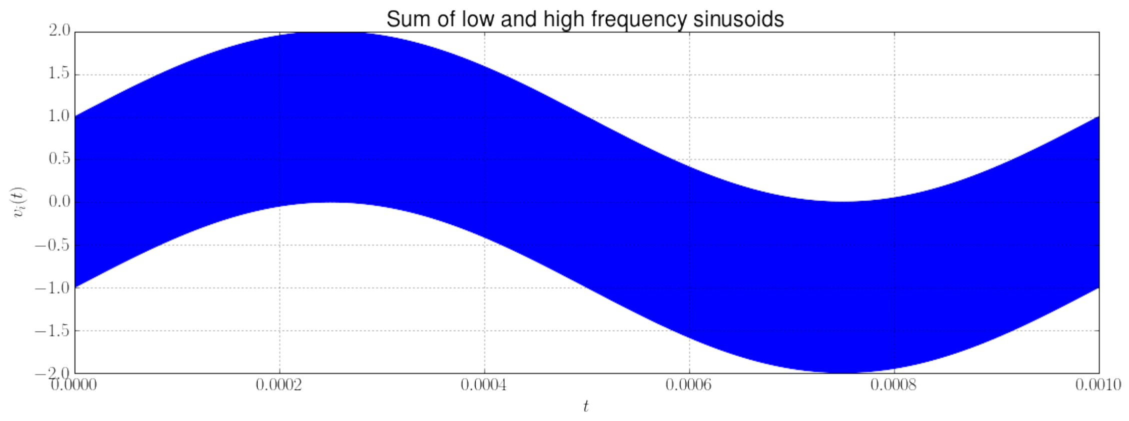
4.2 Sum of low and high frequency sinusoids

We analyse the responses to the following input:

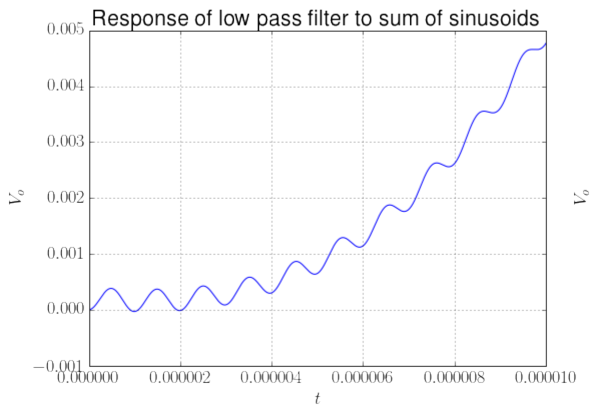
$$v_i(t) = (\sin(2000\pi t) + \cos(2 \times 10^6 \pi t))u(t)$$

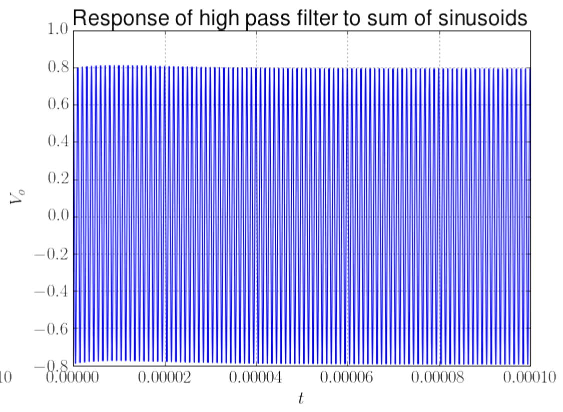
```
def vi1(t):
    """Sum of low frequency and high frequency sinusoids"""
    u_t = 1*(t>0)
    return (np.sin(2000*np.pi*t)+np.cos(2e6*np.pi*t)) * u_t
```

The input waveform is plotted below:



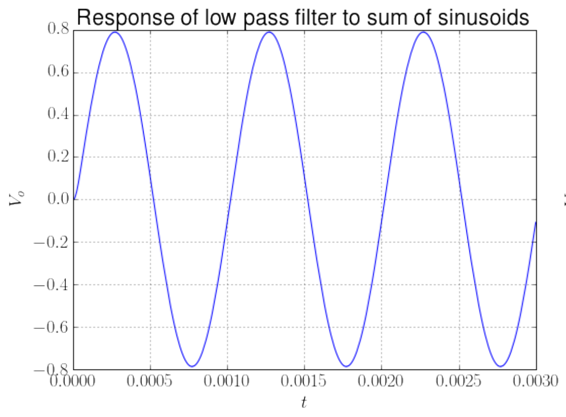
We first plot the response of the low pass filter in a very short time range and the high pass filter in a very large time range.

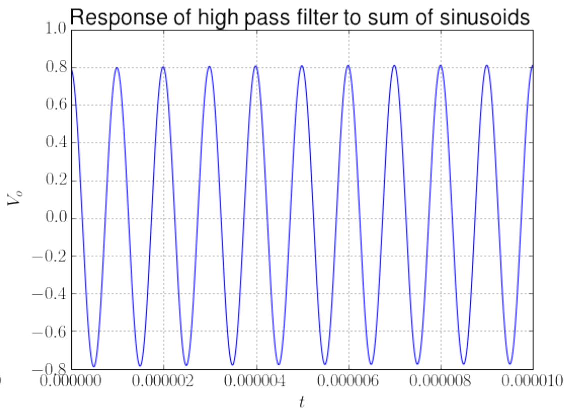




- These time ranges allow us to notice the heavily attenuated part of the input better. We can see that an extremely low amplitude high frequency sinusoid rides on top of a slow response in the output of the low pass filter. This is because the low pass filter has heavily attentuated the high frequency component.
- Similaraly, in the high pass filter, a heavily attenuated low frequency sinusoid modulates a high frequency sinusoid. This is because the high pass filter highly attenuated the low frequency component.

Let us plot the low pass filter response on a larger time scale and the high pass filter response on a smaller time scale:





- We observe that the low pass response more or less looks like a pure sinusoid of the lower frequency with the high frequency completely attenuated.
- We observe that the high pass response also looks like a pure sinusoid of the higher frequency with the low frequency completely attenuated.

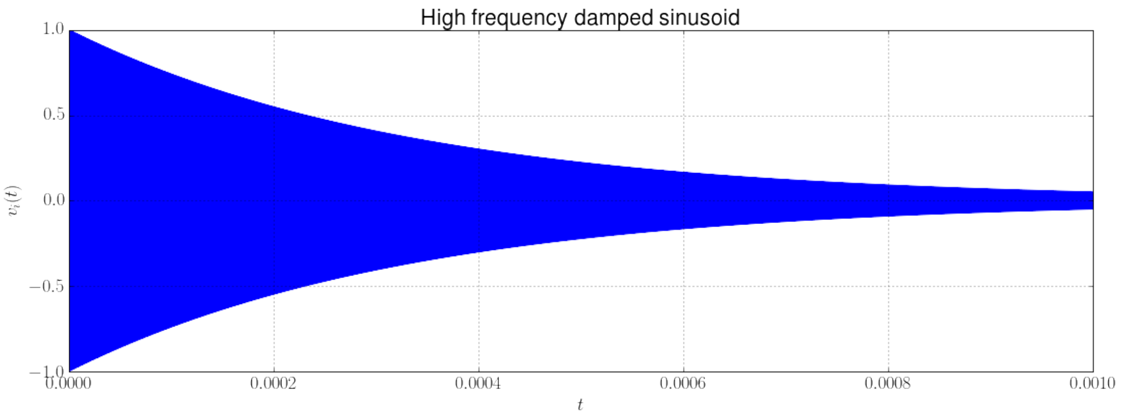
4.3 High frequency damped sinusoid

We analyse the outputs to a high frequency damped sinusoid given as:

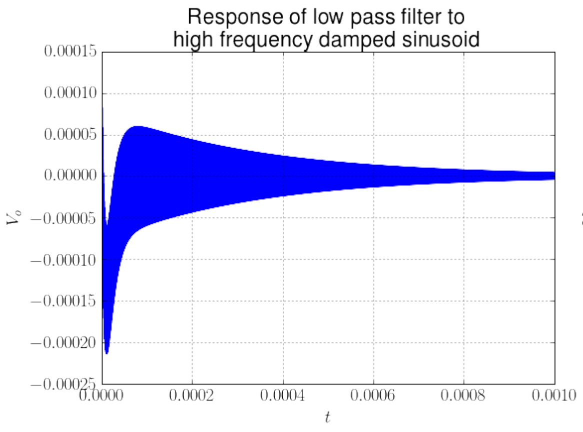
$$v_i(t) = \cos(10^7 t)e^{-3000t}u(t)$$

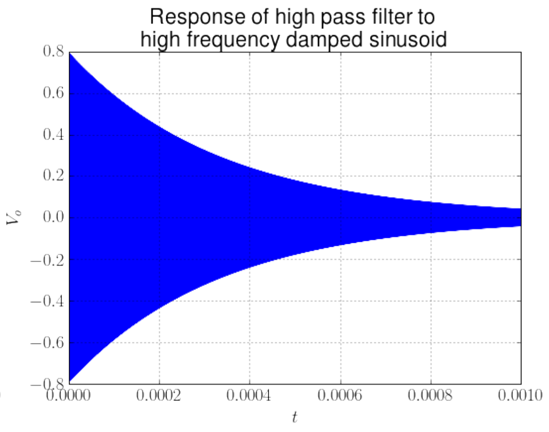
```
def input_f(t,decay=0.5,freq=1.5):
    """Exponentially decaying cosine function."""
    u_t = 1*(t>0)
    return np.cos(freq*t)*np.exp(-decay*t) * u_t
```

The input waveform is plotted below:



The responses are plotted below:





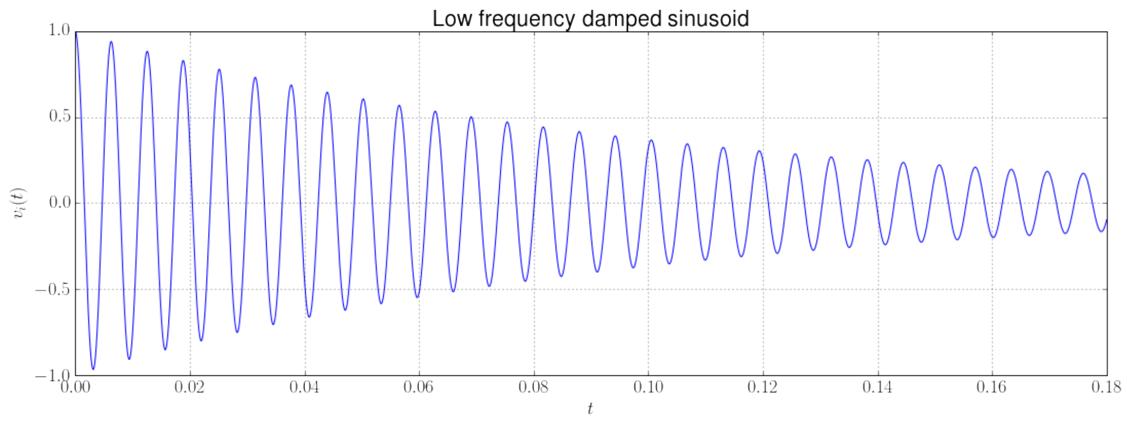
- The low pass filter responds fast and heavily attenuates the high frequency sinusoid. The output decays as the input also decays.
- The high pass filter responds by more or less letting the input pass through as is, with only a slight attenuation. So the output decays as the input does.

4.4 Low frequency damped sinusoid

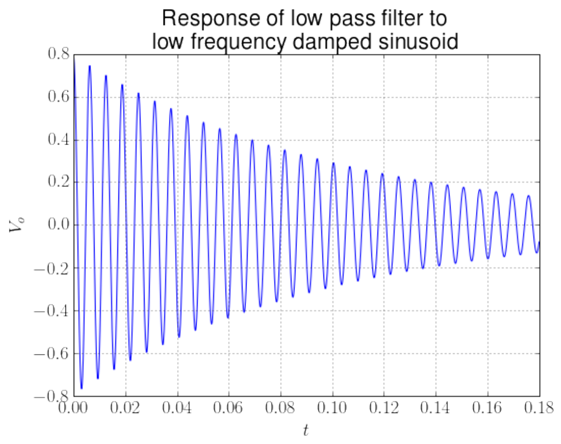
We analyse the outputs to a high frequency damped sinusoid given as:

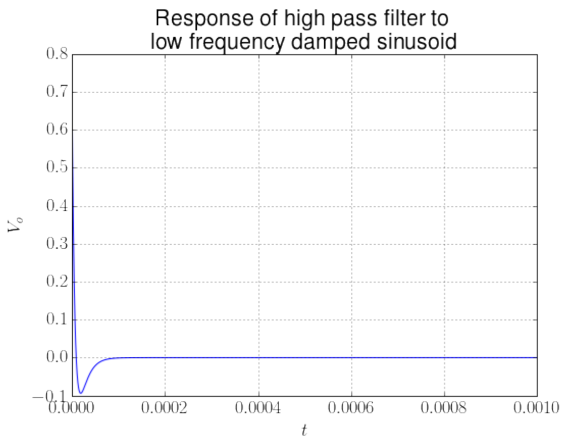
$$v_i(t) = \cos(10^3 t)e^{-10t}u(t)$$

The input waveform is plotted below:



The responses are plotted below:





5 Conclusions

- The low pass filter responds by letting the low frequency sinusoid pass through without much additional attenuation. The output decays as the input also decays.
- The high pass filter responds by quickly attenuating the input. Notice that the time scales show that the high pass filter response is orders of magnitudes faster than the low pass response. This is because the input frequency is below the cutoff frequency, so the output goes to 0 very fast.

- In conclusion, the sympy module has allowed us to analyse quite complicated circuits by analytically solving their node equations. We then interpreted the solutions by plotting time domain responses using the signals toolbox. Thus, sympy combined with the scipy.signal module is a very useful toolbox for analyzing complicated systems like the active filters in this assignment.