# EE2703 : Applied Programming Lab
# A7 Assignment

Keerthana Rachuri
EE20B102

April 13, 2022

# LOW PASS FILTER

Following is the code that declares the low pass function and solves the matrix equation to get the V matrix is as shown below:

Declaring the sympy function for a lowpass filte

```
# Declaring the sympy function for a lowpass filter.
def lowpass(R1,R2,C1,C2,G,Vi):
s = symbols('s')
# Creating the matrices and solving them to get the output voltage.

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
```

Magnitude of transfer function:

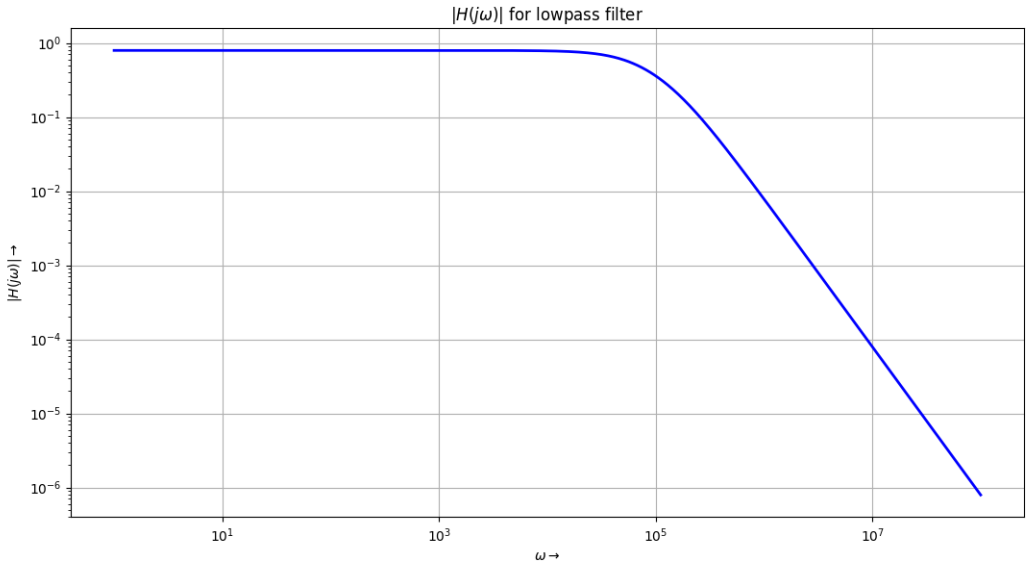


Figure 1:

# High Pass Filter

Following is the code that declares the high pass function and solves the matrix equation to get the V matrix is as shown below:

```
# Declaring the sympy function for a highpass filter.
def highpass(R1,R3,C1,C2,G,Vi):
    s = symbols("s")
# Creating the matrices and solving them to get the output voltage.

    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
    return A,b,V
```

Magnitude of transfer function:

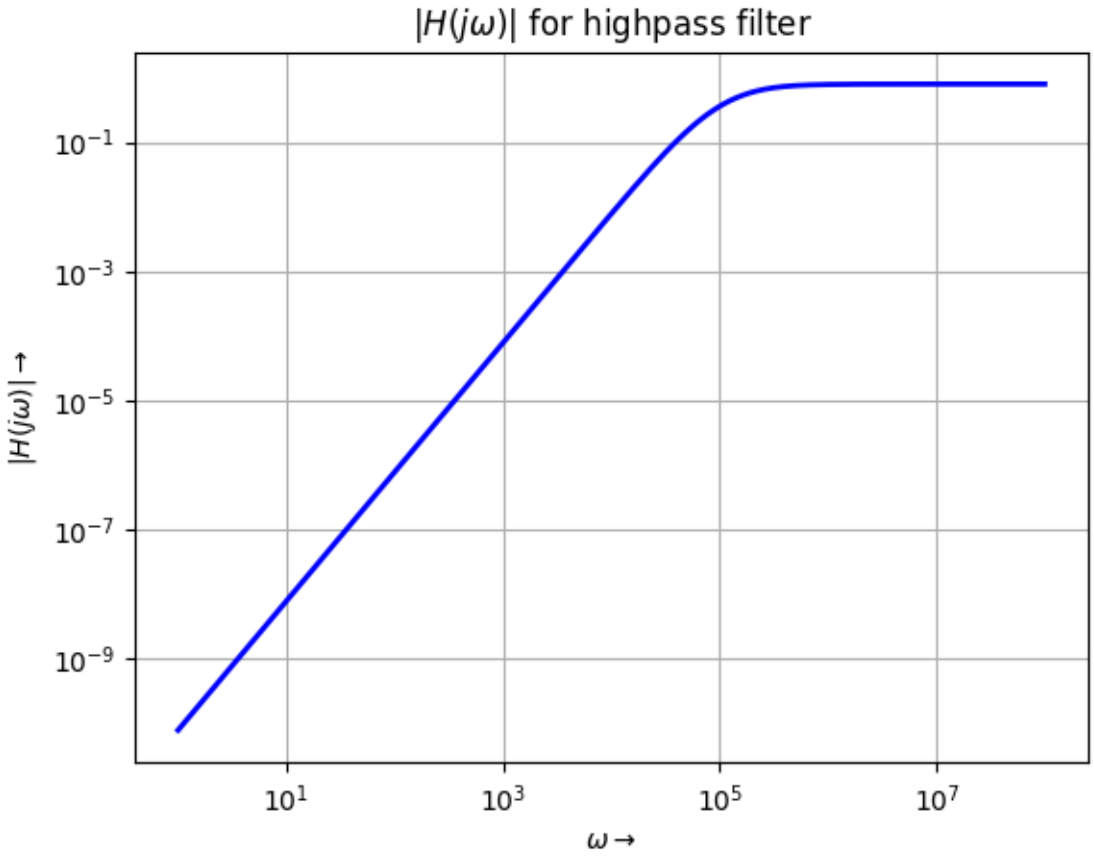


Figure 2:

## Format Change of Sympy Functions

```
# Creating a function to convert a sympy function into a version
that is understood by sp.signal
def sympytoscipy(Y):
n,d=fraction(simplify(Y))
num,den = (np.array(Poly(n,s).all_coeffs(),dtype=float),np.array(Poly(d,s).
    all_coeffs(),dtype=float))
H = sp.lti(num,den)
return H
```

## Step Response Of the Low Pass Filter

In order to find the step response of the low pass filter, we need to assign V i(s) = 1/s. The python code below explains it.

```
# step response for the lowpass filter.
A1,b1,V1 = lowpass(10000,10000,1e-9,1e-9,1.586,1/s)
Vo1 = V1[3]
H1 = sympytoscipy(Vo1)
t,y1 = sp.impulse(H1,None,linspace(0,5e-3,10000))

# Plot for step response of a lowpass filter.
figure(1)
plot(t,y1,'b')
title(r"Step Response for low pass filter")
xlabel(r'$t\rightarrow$')
ylabel(r'$V_o(t)\rightarrow$')
grid(True)
```

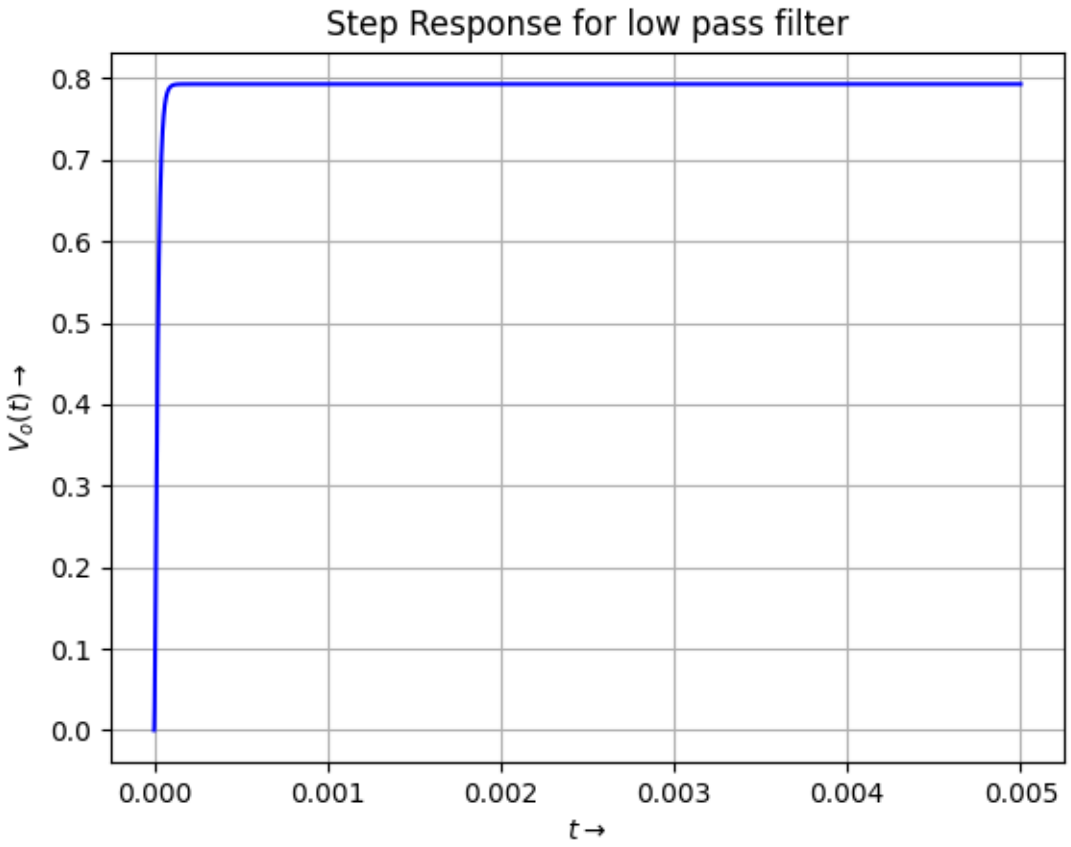


Figure 3:

## Response to Sum Of Sinusoids

The output response for the lowpass filter can easily be found as shown in the following code.

```
# The below piece of code will calculate the transfer function
for the lowpass filter.
s = symbols('s')
A,b,V = lowpass(10000,10000,1e-9,1e-9,1.586,1)
Vo = V[3]
H = sympytoscipy(Vo)
ww = logspace(0,8,801)
ss = 1j*ww
```

```
hf = lambdify(s,Vo,'numpy')
v = hf(ss)

# response for sum of sinusoids.
vi = sin(2000*pi*t) + cos(2e6*pi*t)
t,y2,svec = sp.lsim(H,vi,t)

# The plot for output response for sum of sinusoids of a lowpass filter.
figure(2)
plot(t,y2,'b')
title(r"Output voltage for sum of sinusoids")
xlabel(r'$t\rightarrow$')
ylabel(r'$V_o(t)\rightarrow$')
grid(True)
```

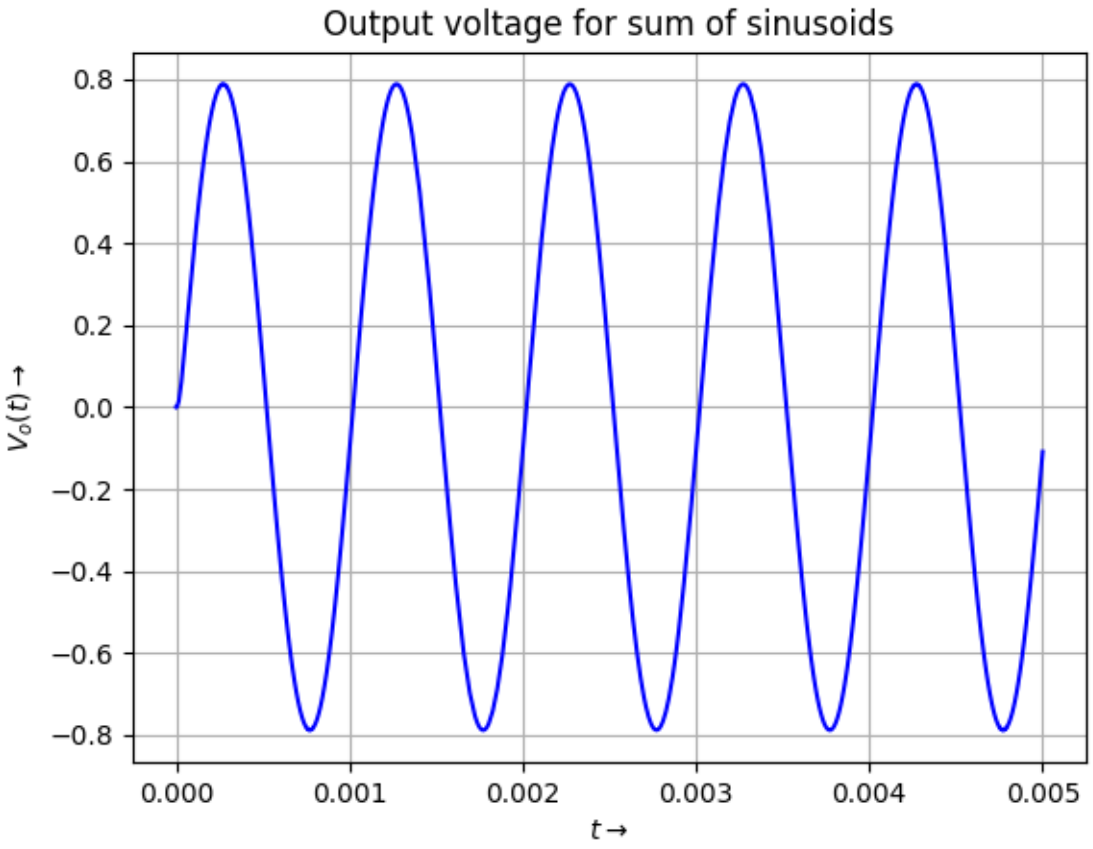
Output voltage for sum of sinusoids
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
$V_o(t) \rightarrow$
0.000
0.001
0.002
0.003
0.004
0.005
$t \rightarrow$

Figure 4:

# Response to Damped Sinusoids

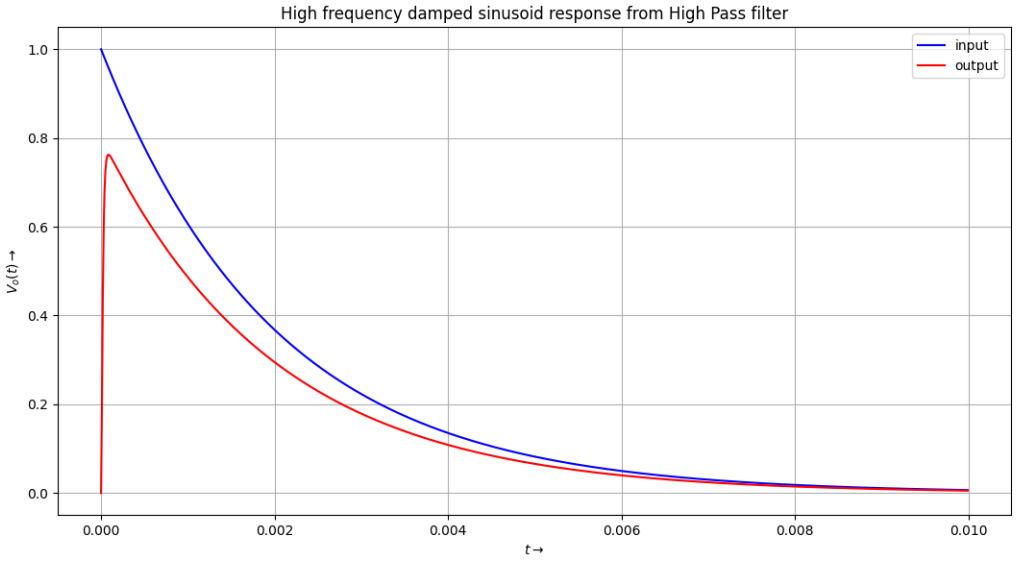


Figure 5:

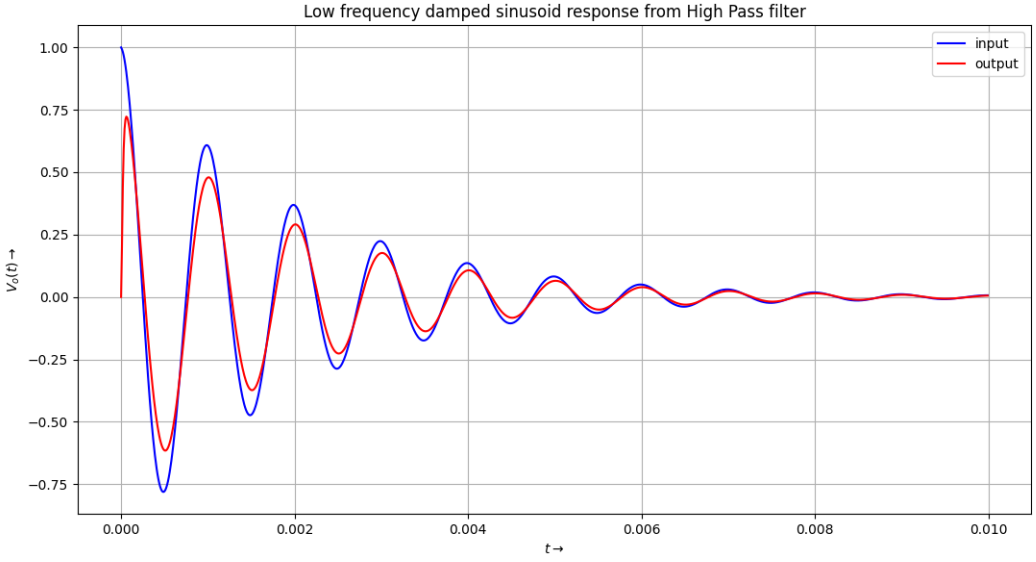


Figure 6:

the python code to execute the above is as shown:

```
# High frequency damped sinusoid.
t2 = arange(0,1e-2,1e-5)
vi_d_1 = exp(-500*t2)*cos(2e6*pi*t2)
t2,y4,svec = sp.lsim(H3,vi_d_1,t2)

# The plot for high frequency damped sinusoid response from highpass filter.
figure(4)
plot(t2,vi_d_1,label='input',color='b')
plot(t2,y4,label='output',color='r')
title(r"High frequency damped sinusoid response from High Pass filter")
xlabel(r'$t\rightarrow$')
ylabel(r'$V_o(t)\rightarrow$')
grid(True)
legend()

# Low frequency damped sinusoid.
t3 = arange(0,1e-2,1e-5)
vi_d_2 = exp(-500*t3)*cos(2e3*pi*t3)
t3,y4_2,svec = sp.lsim(H3,vi_d_2,t3)

# The plot for low frequency damped sinusoid response from highpass filter.
figure(5)
plot(t3,vi_d_2,label='input',color='b')
plot(t3,y4_2,label='output',color='r')
title(r"Low frequency damped sinusoid response from High Pass filter")
xlabel(r'$t\rightarrow$')
ylabel(r'$V_o(t)\rightarrow$')
grid(True)
legend()
```

# Step Response of High Pass Filter

In order to find the step response of the high pass filter, we need to assign V i(s) = 1/s. The python code below explains it.

```
# step response for a highpass filter.
A5,b5,V5 = highpass(10000,10000,1e-9,1e-9,1.586,1/s)
Vo5 = V5[3]
H5 = sympytoscipy(Vo5)
```

```
t5,y5 = sp.impulse(H5,None,linspace(0,5e-3,10000))

# The plot for step response of a highpass filter.
figure(6)
plot(t5,y5,'b')
title(r"Step Response for high pass filter")
xlabel(r'$t\rightarrow$')
ylabel(r'$V_o(t)\rightarrow$')
grid(True)

show()
```

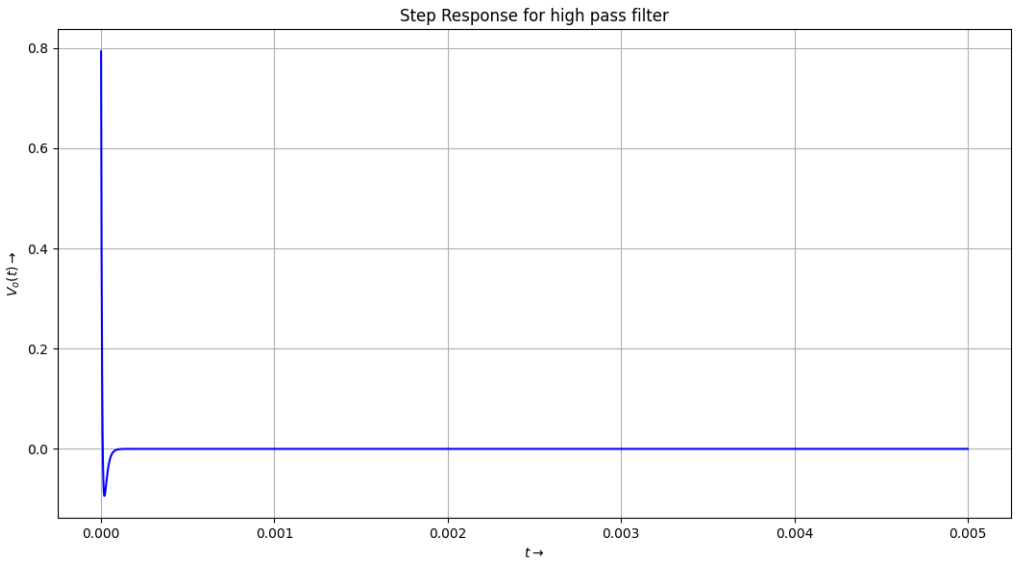


Figure 7:

# Conclusion

Finally, the sympy module has allowed us to analyse rather complex circuits by solving their node equations analytically. Using the signals toolbox, we plotted time domain responses to interpret the results. As a result, sympy in conjunction with the scipy.signal module is a very handy toolkit for evaluating complex systems, such as the active filters in this project.