EE - 2703 Applied Programming Lab
Assignment -8

Analysis of Circuits using Symbolic Python

Mohammed Khandwawala EE16B117

Contents

1 Introduction

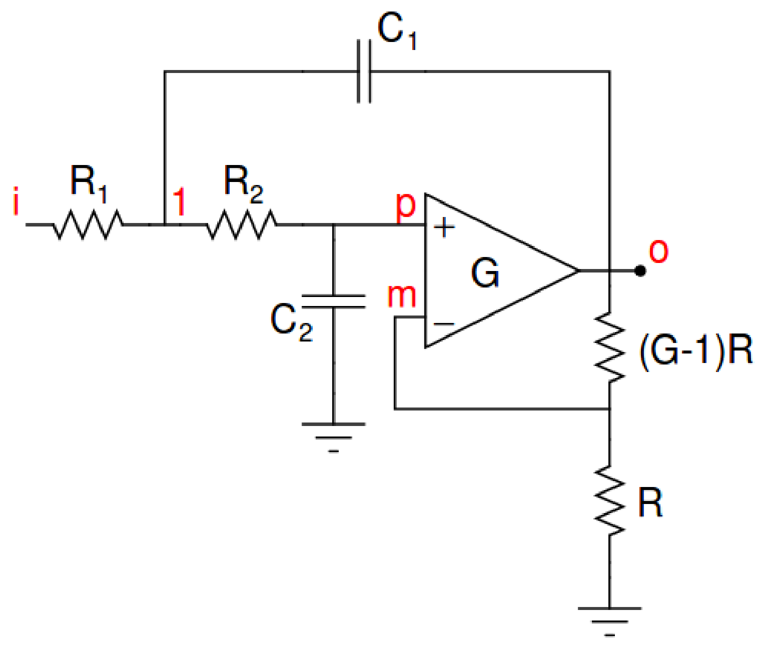
This assignment consists of two circuits . One of a Low-Pass filter and a High-Pass filter. The transfer function of the filters from their circuit is obtained by using Symbolic python package. Writing Modified Nodal Analysis Matrix equation to obtain Transfer Function by taking output voltage to input voltage ratio. From the obtained transfer function various analysis in carried out in this assignment.

Some Python functions that are used in this assignment are:

- **sym.lambdify**("*symbolic variable name*","*symbolic expression*") - This function converts symbolic expression into lambda function
- **sym.simplify**("*symbolic expression*") - This function takes the symbolic expression and return a simplified expression , if not simplified.
- **sym.fraction**("*simplified symbolic expression*") - This returns numerator and denominator of the simplified symbolic expression.
- **sym.poly**("*"Symbolic expression","symbol"*") - This function takes symbolic expression as input and the symbol/s in the expression and converts it in polynomial.
- **polynomial.coeffs()** - This returns the coefficients of the polynomial expression (does not mention the degree)

2 Low-Pass Filter

The Modified Nodal Analysis matrix obtained on solving the first circuit is given by -



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

2.1 Magnitude Response of Low-Pass Filter

Defining the function to solve the above matrix equation to obtain the Transfer Function (Ratio of output to input).

```
def LOWPASS(R1, R2, C1, C2, G, Vi):
    s = symbols('s')
    #Matrix Equations are obtained using modified nodal analysis
    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)
```

Taking R$_1$ = 10KΩ , R$_2$ = 10KΩ, C$_1$ = 1nF , C$_2$ = 1nF , G = 1.586 and Vi as 1. After obtaining transfer function as a symbolic expression (For Vi = 1 , V$_o$ = H(s)) , it is passed to lambdify to get a function that can take return value of the expression. Substituting s by jω and sweeping omega logarithmically from 0 to 10^8.

$$Vo = \frac{0.0001586}{2.0 * 10^{-14}s^2 + 4.414 * 10^{-9}s + 0.0002}$$

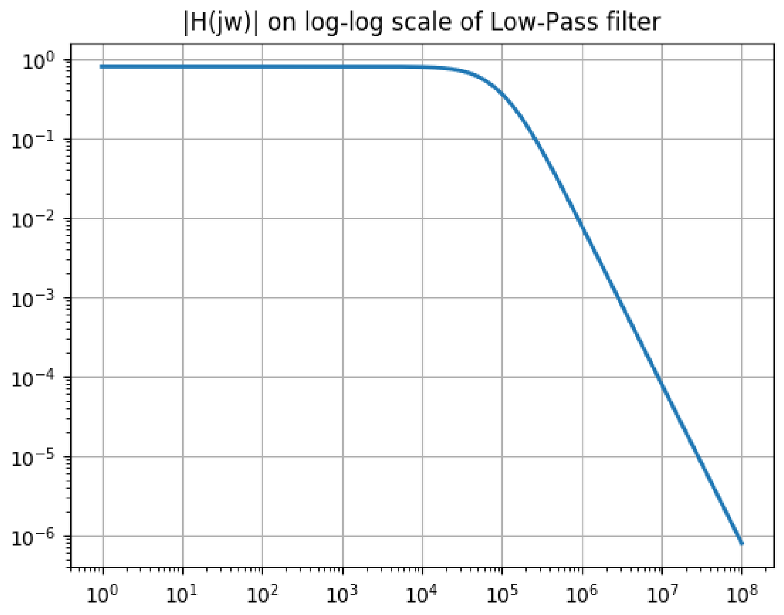
ω_o(Natural Frequency) = 10^5 Rad/sec , Q (Quality Factor) = 0.453 Bandwidth = 56034 Rad/s

```
s = symbols('s')
A,b,V = LOWPASS (10000.0,10000.0,1e−9,1e−9,1.586,1)
Vo = V[3] #obtaining expresion of Vout(symbolic)
print Vo
w = p.logspace(0,8,801) #logarithmic range from 0 to 10**8
hf = lambdify(s,Vo,"numpy") #converts symbolic expression to lambda function
t=np.linspace(0,50,1000) #simulating for 50s
ss = 1j*w
v = hf(ss) #substituting s with jw in lambda function equation
```

The above obtained response in plotted on a log-log scale.

```
p.loglog(w,abs(v),lw=2)
p.grid(True)
p. title ("|H(jw)| on log−log scale of Low−Pass filter")
p.show()
```

The Magnitude response plot of the given filter resembles a Low-Pass filter.



2.2 Output of the filter to input sinusoids

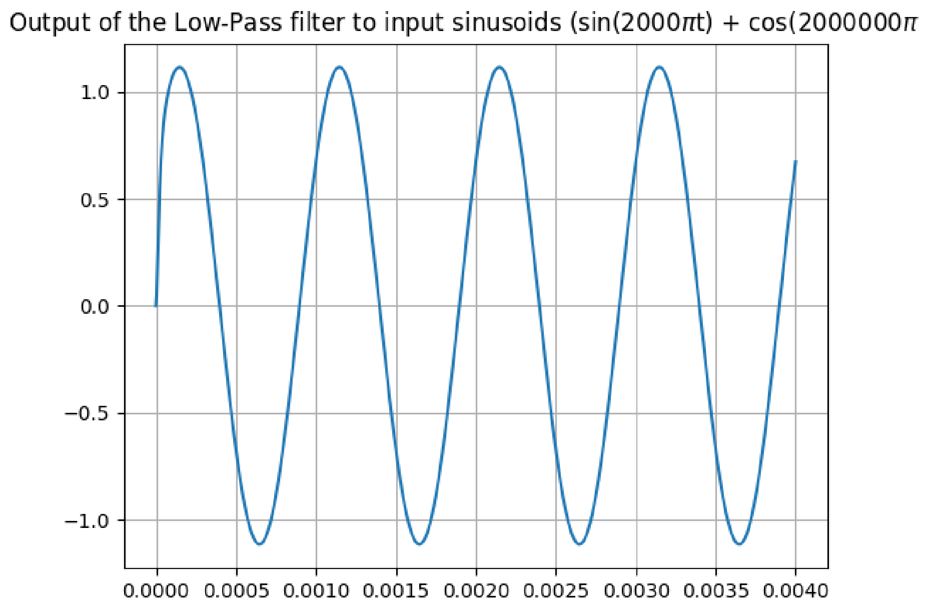
The obtained symbolic expression obtained in the previous part is simplified using simplify function of python (explained earlier). From the simplified transfer function numerator and denominator can be separated using fraction function. Converting the separated symbolic numerator and denominator expression to polynomial using python function Poly and then the coefficients can be extracted to create a polynomial transfer function which can be processed by scipy's signal toolbox.

```
Vo = simplify(Vo) #simplifying transfer function
NUM,DEN = fraction(Vo) #saperating numerator and denominator

print Vo

NUM = poly(NUM,s) #numerator polynomial
DEN = poly(DEN,s) #denominator polynomial

N = list(NUM.coeffs()) #obtaining the coefficients of numerator polynomial in list
D = list(DEN.coeffs()) #obtaining the coefficients of denominator polynomial in list
```

Using signal toolbox function lti we can construct transfer function from the coefficients obtained. For time 3ms we will simulate output for input signal as (sin(2000πt) + cos(2000000πt) using lsim.

```
H = sp.lti([float(N[0])],[float(D[0]),float(D[1]),float(D[2])])
t = np.linspace(0,4*10**−3,1000)
Vi = np.sin(2000*np.pi*t) + np.cos(2*10**6*np.pi*t)
t,V_out,svec = sp.lsim(H,Vi,t)
plt.plot(t,V_out)
plt.grid("True")
plt. title ("Output of the Low−Pass filter to input sinusoids (sin(2000$\pi$t) + cos(2000000$\pi$t))")
plt.show()
```

The output below clearly shows low-pass response of the filter . Only sinusoid with 1000Hz is visible in the output . Frequency of the sinusoid can be measured from the obtained plot.

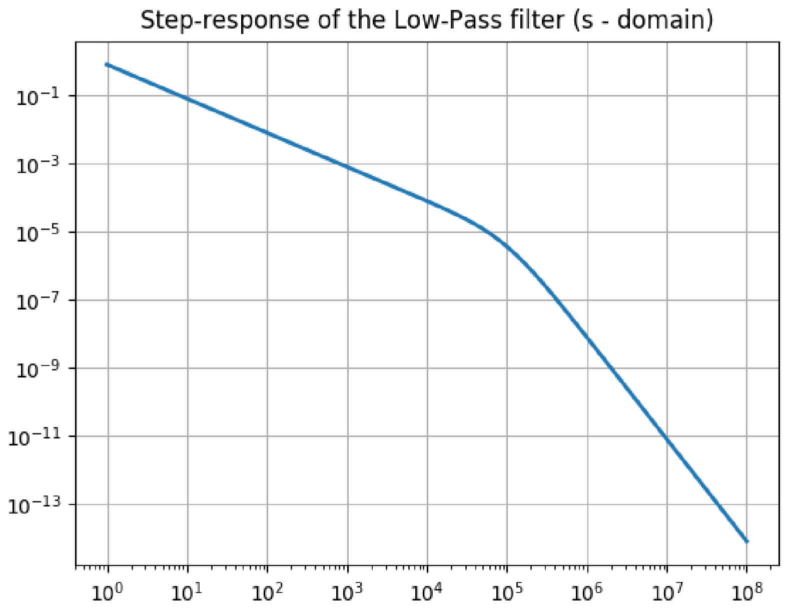


2.3 Step Response of a Low Pass Filter

Step Response of system is output to u(t). We have the Transfer Function of the Low pass Filter. So instead of solving for input 1 V . Already defined function has Vin as a parameter to which $\frac{1}{s}$ can be passed instead on 1.

```
A,b,V = LOWPASS (10000.0,10000.0,1e−9,1e−9,1.586,1/s)
Vo = V[3]
w = p.logspace(0,8,801)
hf = lambdify(s,Vo,"numpy")
t=np.linspace(0,50,1000)
ss = 1j*w
v = hf(ss)
p.loglog(w,abs(v),lw=2)
p.grid(True)
p.title("Step−response of the Low−Pass filter (s − domain)")
p.show()
```

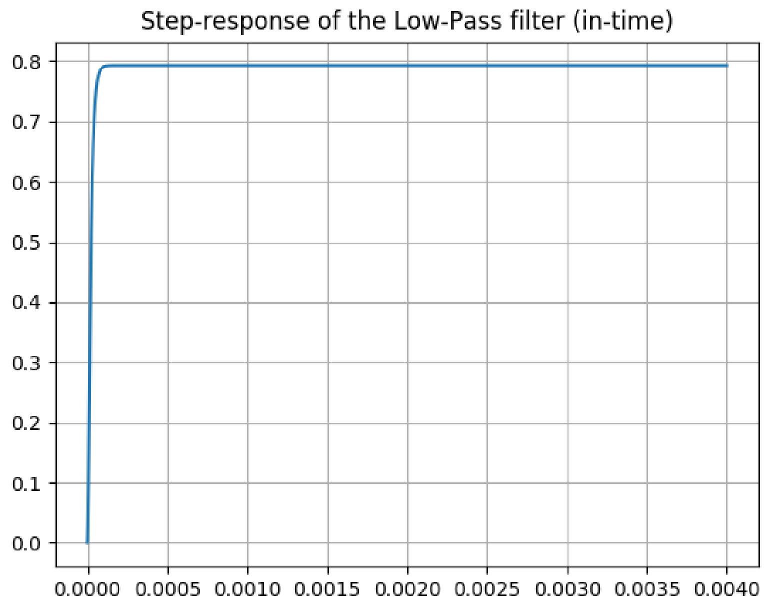
The obtained s domain response is shown below.



To obtain the step response in time domain, already obtained polynomial transfer function can be passed u(t) (input as 1 for all t>0) using lsim function .

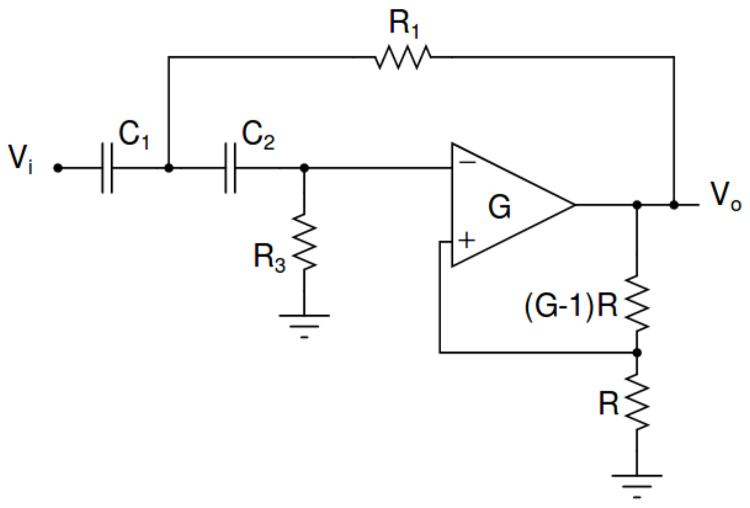
```
H = sp.lti ([float(N [0])],[ float(D[0]),float(D[1]),float(D[2])])
t  = np.linspace(0,4*10**-3,100000)
Vi = np.ones(100000)
t,V_out,svec = sp.lsim(H,Vi,t)
plt. title ("Step-response of the Low-Pass filter (in-time)")
plt.plot(t,V_out)
plt.grid(True)
plt.show()
```

Time domain response to step is shown below in the figure. It falls flat without ripples because the quality factor of the low pass filter is very low.



3 High-Pass Filter

The Modified Nodal Analysis matrix obtained on solving the first circuit is given by -



$$\begin{bmatrix} 0 & 0 & 1 & \frac{-1}{G} \\ 0 & -G & G & 1 \\ \frac{-sC_2}{1+sR_2C_2} & \frac{1}{R_3} & 0 & 0 \\ \frac{1}{R_1}+sC_1+sC_2 & -sC_2 & 0 & \frac{-1}{R_1} \end{bmatrix} \begin{bmatrix} V_1 \\ V_p \\ V_m \\ V_o \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ C_1V_is \end{bmatrix} \tag{2}$$

3.1 Magnitude Response of High-Pass Filter

Defining the function to solve the above matrix equation to obtain the Transfer Function (Ratio of output to input).

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

Taking R_1 = 10KΩ , R_2 = 10KΩ, C_1 = 1nF , C_2 = 1nF , G = 1.586 and Vi as 1. After obtaining transfer function as a symbolic expression (For Vi = 1 , V_o = H(s)) , it is passed to lambdify to get a function that can take return value of the expression. Substituting s by jω and sweeping omega logarithmically from 0 to 10^8.

$$Vo = \frac{1.586e^{-18} * s^2}{2.0 * 10^{-18}s^2 + 4.414 * 10^{-13}s + 2 * 10^{-8}}$$

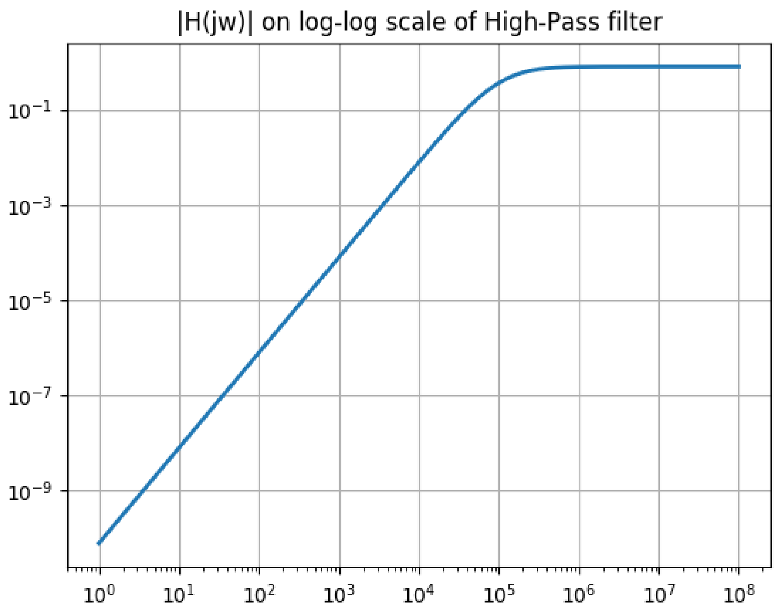
ω_o(Natural Frequency) = 10^5 Rad/sec , Q (Quality Factor) = 0.453 and -3dB bandwidth is 17800 Rad/sec

```
s = symbols('s')
A,b,V = HIGHPASS(10000.0,10000.0,1e−9,1e−9,1.586,1)
Vo = V[3]
w = p.logspace(0,8,801)
hf = lambdify(s,Vo,"numpy")
ss = 1j*w
v1 = hf(ss)
```

The above obtained response in plotted on a log-log scale.

```
p.loglog(w,abs(v1),lw=2)
p.grid(True)
p.title("|H(jw)| on log-log scale of High-Pass filter")
p.show()
```

The Magnitude response plot of the given filter resembles a High-Pass filter.



3.2 Output of the filter to input sinusoids

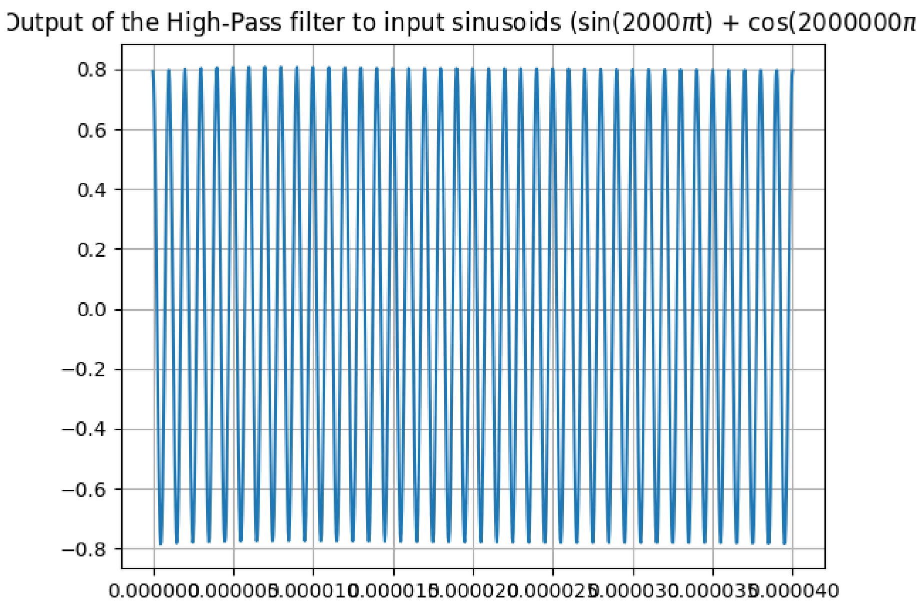
The obtained symbolic expression obtained in the previous part is simplified using simplify function of python (explained earlier). From the simplified transfer function numerator and denominator can be separated using fraction function. Converting the separated symbolic numerator and denominator expression to polynomial using python function Poly and then the coefficients can be extracted to create a polynomial transfer function which can be processed by scipy's signal toolbox.

```
Vo = simplify(Vo) #simplifying transfer function
NUM,DEN = fraction(Vo) #saperating numerator and denominator

print Vo

NUM = poly(NUM,s) #numerator polynomial
DEN = poly(DEN,s) #denominator polynomial

N = list(NUM.coeffs()) #obtaining the coefficients of numerator polynomial in list
D = list(DEN.coeffs()) #obtaining the coefficients of denominator polynomial in list
```

Using signal toolbox function lti we can construct transfer function from the coefficients obtained. For time 3ms we will simulate output for input signal as (sin(2000πt) + cos(2000000πt) using lsim.

```
H = sp.lti([float(N [0]),0.0,0.0],[ float(D[0]),float(D[1]),float(D[2])])
t = np.linspace(0,4*10**-5,1000)
Vi = np.sin(2000*np.pi*t) + np.cos(2*10**6*np.pi*t)
t,V_out,svec = sp.lsim(H,Vi,t)
plt.plot(t,V_out)
plt.grid(True)
plt.title("Output of the High-Pass filter to input sinusoids (sin(2000$\pi$t) + cos(2000000$\pi$t))")
plt.show()
```

The output below clearly shows High-pass response of the filter . Only sinusoid with 1MHz is visible in the output . Frequency of the sinusoid can be measured from the obtained plot.



3.3 Response of High-Pass filter to damped Sinusoid

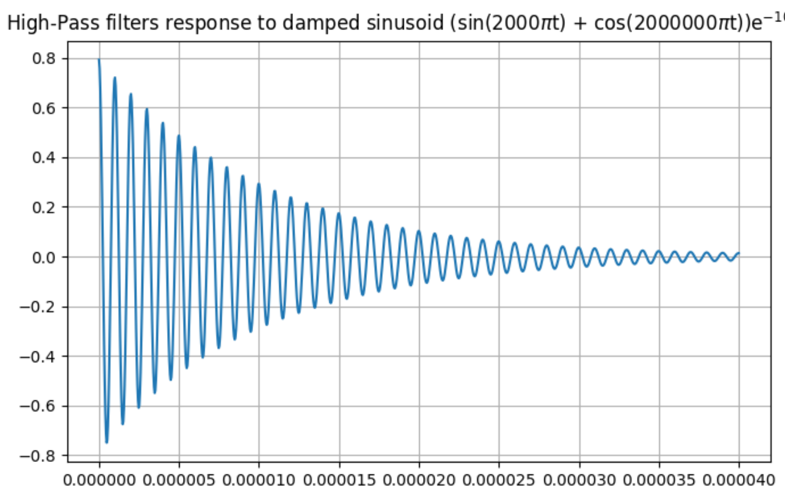
Similar to previous section , the obtained polynomial trasfer function is again used , with the help of python function lsim time domain output to the damped sinusoid is obtained. Sinusoidal input given is

$$[sin(2000\pi t) + cos(2000000\pi t)]e^{-\alpha t}$$

The simulation time is 40μs and α is chosen to be 100000.

```
H = sp.lti([float(N [0]),0.0,0.0],[ float(D[0]),float(D[1]),float(D[2])])
t = np.linspace(0,4*10**-5,1000)
Vi = (np.sin(2000*np.pi*t) + np.cos(2*10**6*np.pi*t))*np.exp(-100000*t)
t,V_out,svec = sp.lsim(H,Vi,t)
plt.plot(t,V_out)
plt.title("High-Pass filters response to damped sinusoid....
...( sin(2000$\pi$t) + cos(2000000$\pi$t))e$^{-100000t}$")
plt.grid(True)
plt.show()
```

The Output of the damped sinusoid to the High Pass filter has a decaying sinusoid of frequency 1MHz ,as the lower frequency components gets highly attenuated.

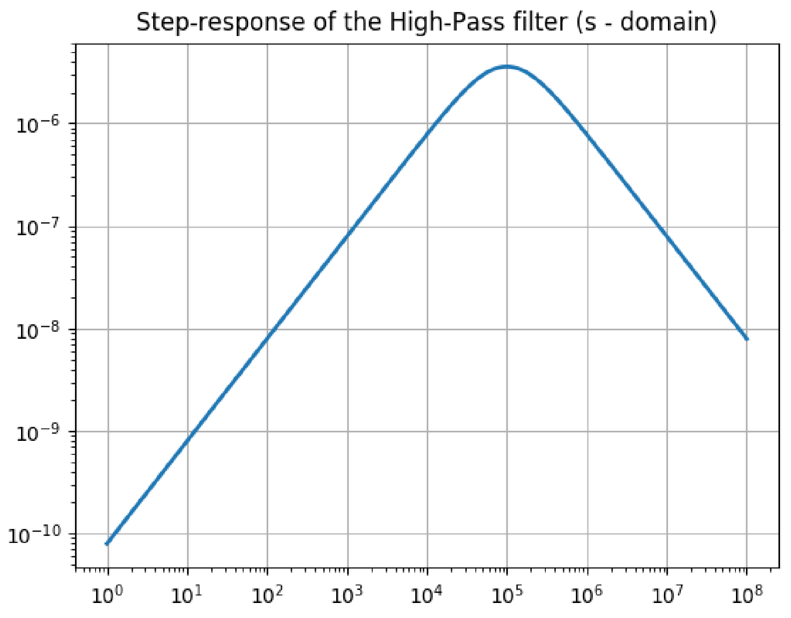


3.4 Step Response of a High-Pass Filter

Step Response of system is output to u(t). We have the Transfer Function of the High pass Filter. So instead of solving for input 1 V . Already defined function has Vin as a parameter to which $\frac{1}{s}$ can be passed instead on 1.

```
A,b,V = HIGHPASS(10000.0,10000.0,1e−9,1e−9,1.586,1/s)
Vo = V[3]
w = p.logspace(0,8,801)
hf = lambdify(s,Vo,"numpy")
ss = 1j*w
v1 = hf(ss)
p.loglog(w,abs(v1),lw=2)
p.grid(True)
p.title("Step−response of the High−Pass filter (s − domain)")
p.show()
```

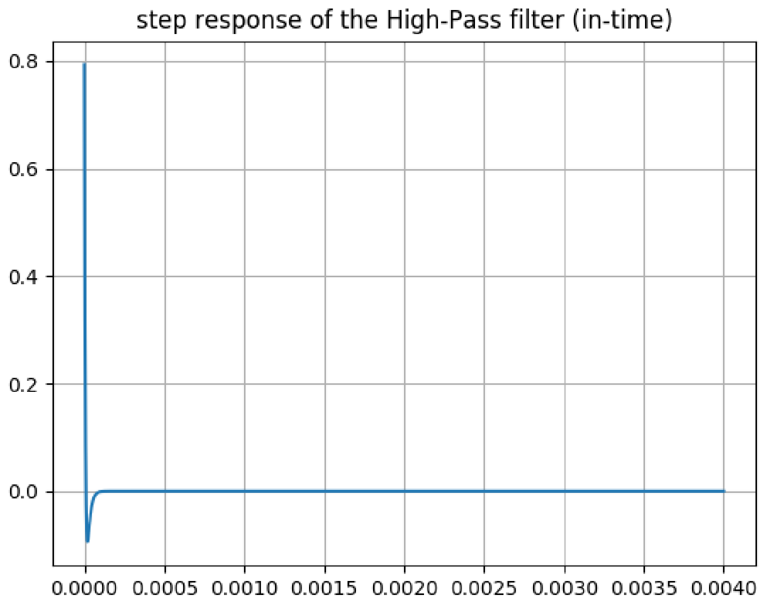
The obtained s domain response is shown below. It is a band pass filter as integrating a High pass filter gives a band-pass filter.



To obtain the step response in time domain, already obtained polynomial transfer function can be passed u(t) (input as 1 for all t>0) using lsim function .

```
H = sp.lti([float(N [0]),0.0,0.0],[ float(D[0]),float(D[1]),float(D[2])])
t = np.linspace(0,4*10**-3,100000)
Vi = np.ones(100000)
t,V_out,svec = sp.lsim(H,Vi,t)
plt. title ("step response of the High-Pass filter (in-time)")
plt.grid(True)
plt.plot(t,V_out)
plt.show()
```

Time domain response to step is shown below in the figure. It falls flat without ripples because the quality factor of the High pass filter is very low.


step response of the High-Pass filter (in-time)