Assignment_8

April 5, 2018

1 Assignment 8

EE2703: Applied Programming
Author: Varun Sundar, EE16B068

2 Abstract

This week's assignment involves the analysis of active filter's using laplce transforms. Python's symbolic solving library, sympy, comes off as a particularly nifty tool to handle our requirements in solving Modified Nodal Analysis (MNA) equations. Besides this the library also includes useful classes to handle the simulation and response to inputs.

Coupled with *scipy's signal* module, we are able to analyse both Sallen-Key and Raunch filters, two common active filter topologies to attain second order filters with just a single operational amplifier.

3 Introduction

Reference to Horowitz and Hill (Active Filters) : link

We begin by understanding the usage of sympy in defining, evaluating and plotting symbolic equations. Besides this, we also examine the use case of matrix equation for MNA solutions. With this, we start analysing the case of an active low pass filter.

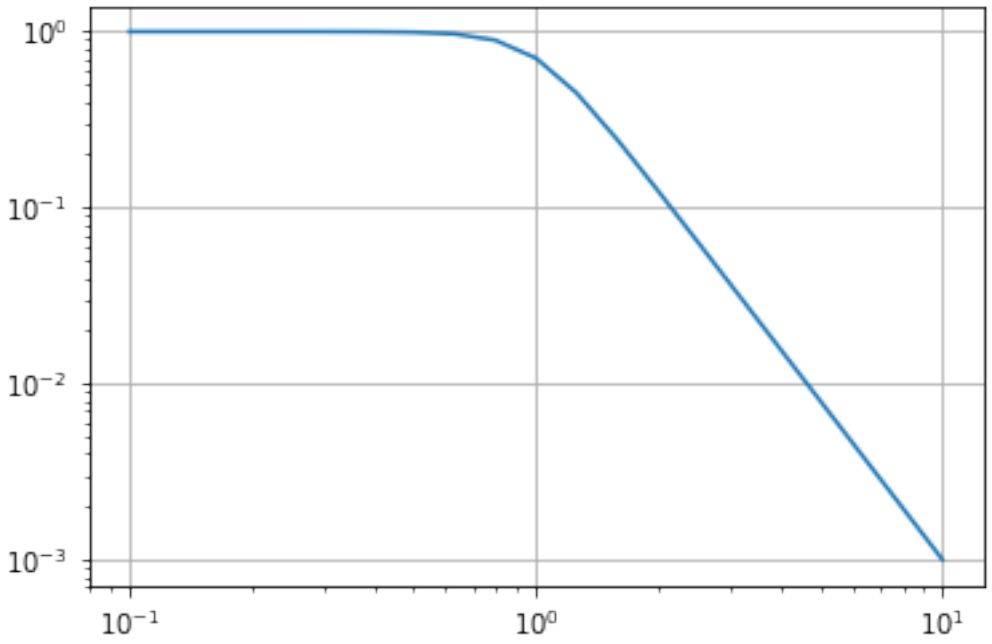
Conventions: 1. We are using Python 3, GCC for C 2. Underscore naming vs Camel Case 3. PEP 25 convention style.

```
In [216]: import sympy
          from IPython.display import display
          import numpy as np
          import matplotlib.pyplot as plt

          import scipy.signal as sp

          # Use either

          sympy.init_session
          #sympy.init_printing(use_unicode=True)
          sympy.init_printing(use_latex='mathjax')
```

Here, we use sympy's lambdify method to plot the Bode Plot of the transfer function,

$$H(s) = 1/(s^3 + 2 * s^2 + 2s + 1)$$

```
In [217]: s=sympy.symbols('s')
          h=1/(s**3+2*s**2+2*s+1)
          w=np.logspace(-1,1,21)
          ss=1j*w
          f=sympy.lambdify(s,h,"numpy")

          plt.loglog(w,np.abs(f(ss)))
          plt.grid(True)
          plt.show()
```



We solve for an active low pass filter (Sallen-Key). This particular active filter is known for being a single op-amp based low pass filter.

The MNA (Nodal) Equations may be recast as:

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

Note that the lab assignment sheet used,

$$V_o = G(V_p V_m)$$

incorrectly, since the op-amp is a large gain device. This should be replaced by:

$$V_o = G(V_p)$$

which correctly replaces the intended gain block.

```
In [218]: def lowpass(R1,R2,C1,C2,G,Vi):
              s=sympy.symbols('s')
              A=sympy.Matrix([[0,0,1,-1/G],[-1/(1+s*R2*C2),1,0,0], \
              [0,-G,0,1],[-1/R1-1/R2-s*C1,1/R2,0,s*C1]])
              b=sympy.Matrix([0,0,0,Vi/R1])
              V=A.inv()*b
              return (A,b,V)
```

```
In [219]: A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)

          print ('G=1000')
          Vo=V[3]
          # Computed inverse
          print ("Computed Inverse")
          display(Vo)
          # Simplify existing expression
          print ("Simplified Expression")
          Vo=sympy.simplify(Vo)
          display(Vo)

          s=sympy.symbols('s')
          display((1/Vo).coeff(s))

          a=sympy.Poly(1/Vo,s)
          display(a.all_coeffs())

          w=np.logspace(0,8,801)
          ss=1j*w
          hf=sympy.lambdify(s,Vo,"numpy")
          v=hf(ss)

          plt.loglog(w,abs(v),lw=2)
          plt.grid(True)
          plt.show()
```

```
G=1000
Computed Inverse
```

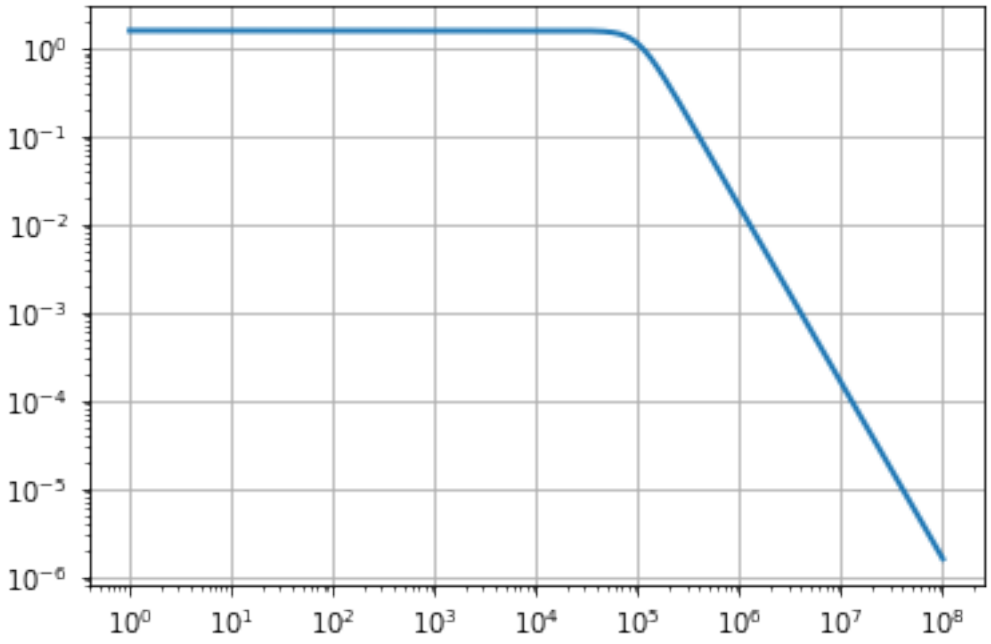
$$\frac{0.0001586}{(1.0 \cdot 10^{-5} s + 1)\left(-1.0 \cdot 10^{-9} s + \frac{1.586 \cdot 10^{-9} s}{1.0 \cdot 10^{-5} s + 1} - 0.0002 + \frac{0.0001}{1.0 \cdot 10^{-5} s + 1}\right)}$$

```
Simplified Expression
```

$$-\frac{0.0001586}{1.0 \cdot 10^{-14} s^2 + 1.414 \cdot 10^{-9} s + 0.0001}$$

$$-8.91551071878941 \cdot 10^{-6}$$

$$\left[-6.30517023959647 \cdot 10^{-11}, \quad -8.91551071878941 \cdot 10^{-6}, \quad -0.630517023959647\right]$$



4 Assignment Questions

4.1 Question 1

To obtain the step response of the circuit, we do so in the laplace domain and then convert it to the time domain.

```
In [220]: H=Vo*1/s
          display(H)
          H=sympy.simplify(H)
          print (" Simplified Expression ")
          display(H)
```

$$-\frac{0.0001586}{s\left(1.0 \cdot 10^{-14} s^2 + 1.414 \cdot 10^{-9} s + 0.0001\right)}$$

Simplified Expression

$$-\frac{0.0001586}{s\left(1.0\cdot 10^{-14}s^2+1.414\cdot 10^{-9}s+0.0001\right)}$$

We compute the time-domain response of the active low pass filter.

In [237]:
```
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)

        print ('G=1000')
        Vo=V[3]
        Vo=sympy.simplify(Vo)
        display(Vo)

        s,t=sympy.symbols("s t")
        t=sympy.Symbol("t",positive=True)
        n,d = sympy.fraction(Vo)
        n_sp,d_sp=(np.array(sympy.Poly(j,s).all_coeffs(),dtype=float) for j in (n,d))

        print(n_sp,d_sp)
        ts=np.linspace(0,0.001,8001)
        t,x,svec=sp.lsim(sp.lti(n_sp,d_sp),np.ones(len(ts)),ts)
        # Plot the absolute step response
        plt.plot(t,np.abs(x),lw=2)
        plt.grid(True)
        plt.show()
```

```
G=1000
```

$$-\frac{0.0001586}{1.0\cdot 10^{-14}s^2+1.414\cdot 10^{-9}s+0.0001}$$

```
[-0.0001586] [  1.00000000e-14   1.41400000e-09   1.00000000e-04]
```