

Python Assignment No. 7

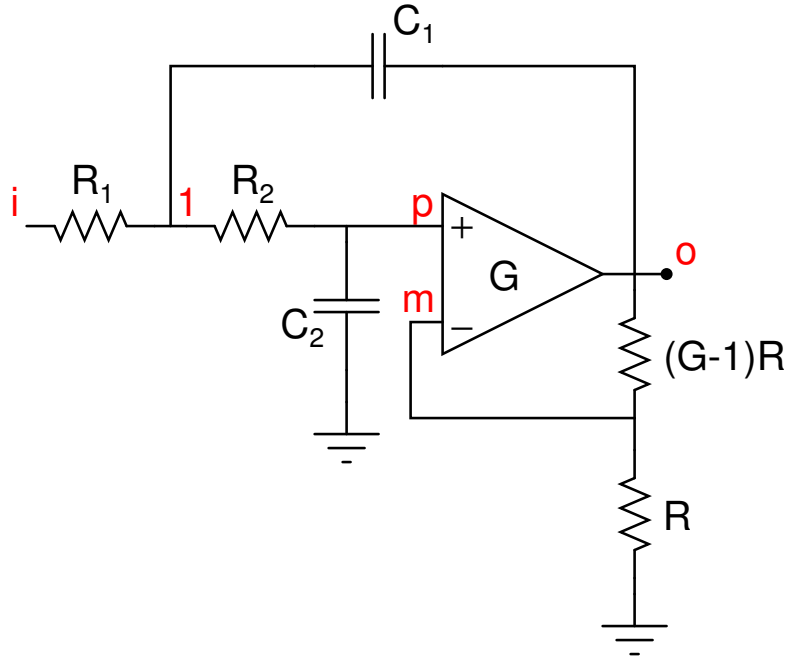
March 12, 2019

In this assignment, the focus will be on two powerful capabilities of Python:

- Symbolic Algebra
- Analysis of Circuits using Laplace Transforms

1 Analysis of Circuits using Laplace Transforms

Consider the following figure (from page 274 of Horowitz and Hill):



where $G = 1.586$ and $R_1 = R_2 = 10k\Omega$ and $C_1 = C_2 = 10pF$. This gives a 3dB Butterworth filter with cutoff frequency of $1/2\pi MHz$.

The circuit equations are

$$V_m = \frac{V_o}{G} \quad (1)$$

$$V_p = V_1 \frac{1}{1 + j\omega R_2 C_2} \quad (2)$$

$$V_o = G(V_p - V_m) \quad (3)$$

$$\frac{V_i - V_1}{R_1} + \frac{V_p - V_1}{R_2} + j\omega C_1(V_o - V_1) = 0 \quad (4)$$

Solving for V_o in 3, we get

$$V_o = \frac{GV_1}{2} \frac{1}{1 + j\omega R_2 C_2}$$

Thus, Eq. 4 becomes an equation for V_1 as follows:

$$\frac{V_i}{R_1} + V_1 \left(-\frac{1}{R_1} + \frac{1}{R_2} \frac{1}{1 + j\omega R_2 C_2} - \frac{1}{R_2} + j\omega C_1 \frac{G}{2} \frac{1}{1 + j\omega R_2 C_2} - j\omega C_1 \right) = 0$$

The term involving G will dominate, and so we obtain

$$V_1 \approx \frac{2V_i}{G} \frac{1 + j\omega R_2 C_2}{j\omega R_1 C_1}$$

Substituting back into the expression for V_o we finally get

$$V_o \approx \frac{V_i}{j\omega R_1 C_1} \quad (5)$$

We would like to solve this in Python and also get (and plot) the exact result. For this we need the sympy module.

2 Introduction to SymPy

Start *ipython* and import the sympy module

```
$ ipython
>>> from sympy import *
>>> init_session
>>> x,y,z = symbols('x y speed')
```

Normally we invoke *ipython* and `import *` from *pylab*. Instead we invoke *sympy*. We are now ready to do symbolic work. The last line allows us to define variables that can be used for symbolic manipulations. The string inside the `symbols` command tells *sympy* how to print out expressions involving these symbols. For instance

```
>>> x=y+1/z
>>> x
y + 1/speed
```

This is confusing, so just use the variable name for the symbol. One place where this is useful is for greek symbols:

```
>>> w=symbols('omega')
>>> x=w*y+z
omega*y+z
```

Note that **symbols are not python variables:**

```
>>> expr=x+1
>>> print expr
x + 1 # a symbolic expression
>>> x=2 # reassignment of x
>>> print expr
x + 1 # changing x did not change expr
>>> print x
2 # did change x though
>>> expr=x+1
>>> print expr
3 # now expr used current value of x - no longer a symbol
```

This can lead to confusion, so be careful.

Rational functions of s are straightforward in sympy

```
>>> s=symbols('s R_2 C_2')
>>> h=-1/(1+s*R2*C2)
>>> print h
-1/(C_2*R_2*s+1)
```

We will require to create some matrices. For example

```
>>> M=Matrix([[1 2],[3 4]])
>>> print M
```

creates a 2×2 matrix containing

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

If you define a matrix and a column vector you can multiply them as in Matlab using “*”

```
>>> N=Matrix([5,6])
>>> M*N
Matrix([
[17],
[39]])
```

Suppose you want to solve $Mx = N$. Then we need to use the inverse of M

```
>>> M.inv()*N
Matrix([
[-4],
[9/2]])
```

Finally, to link to the Python we already know, we need to be able to generate numbers out of symbols. For that we use *evalf*.

```
>>> expr=sqrt(8)
>>> print expr
2*sqrt(2)
>>> expr.evalf()
2.82842712474619
```

This works for single numbers. But what if we want to plot the magnitude response of a laplace transform expression? We use something called *lambdify*

```
>>> h=1/(s**3+2*s**2+2*s+1)
>>> import pylab as p
>>> w=p.logspace(-1,1,21)
>>> ss=1j*w
>>> f=lambdify(s,h,"numpy")
>>> p.loglog(w,abs(f(ss)))
>>> p.grid(True)
>>> p.show()
```

Converts "h" into a function "f" of variable "s".

What *lambdify* does is it takes the sympy function “h” and converts it into a python function which is then applied to the array ‘ss’. This is much faster than iterating over the elements of ss and using *evalf* on the elements.

3 Using Sympy to solve our circuit problem

We can solve directly for the exact result from Python. Let us define $s = j\omega$, and rewrite the equations in a matrix equation

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

This is the equation that we create and solve for now. The following function both defines the matrices and solves for the solution.

```
from sympy import *
import pylab as p
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])
```

That defines the coefficient matrices. Note that I did not include $V_i(s)$ in the vector b . I multiply it in later. However, I could have included it here as well. Now we solve for the solution.

```
V=A.inv()*b
```

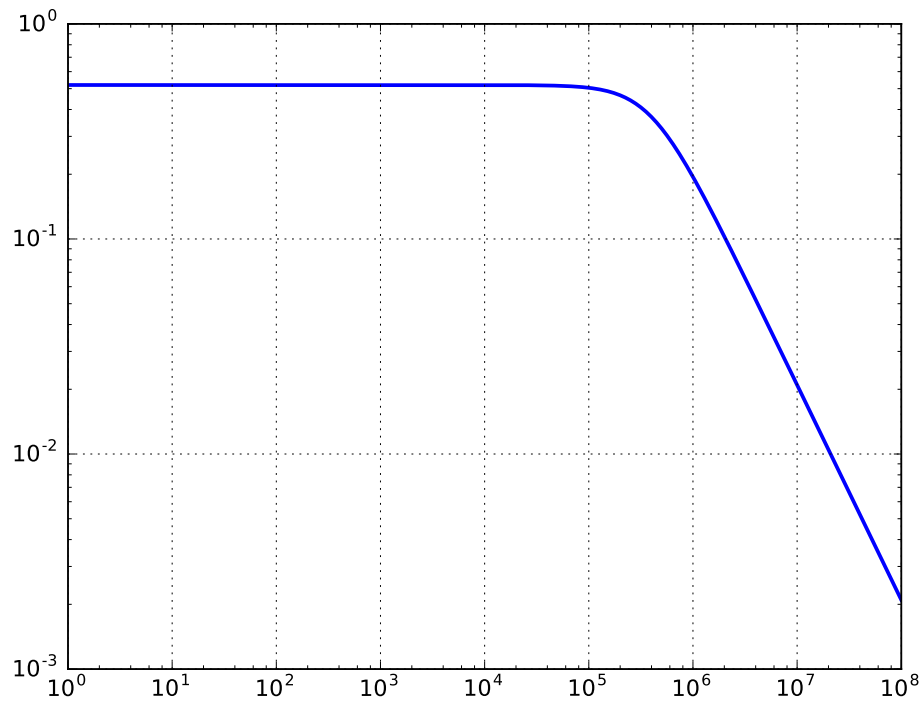
This solution is in s space.

Now extract the output voltage. Note that it is the fourth element of the vector. The logic of this line is explained below the code.

```
return (A,b,V)
```

We now use this function to generate the plot for the values mentioned above. Note that the input voltage is a unit step function.

```
A,b,V=lowpass(10000,10000,1e-9,1e-9,1.586,1)
print 'G=1000'
Vo=V[3]
print Vo
w=p.logspace(0,8,801)
ss=1j*w
hf=lambdify(s,Vo,'numpy')
v=hf(ss)
p.loglog(w,abs(v),lw=2)
p.grid(True)
p.show()
```



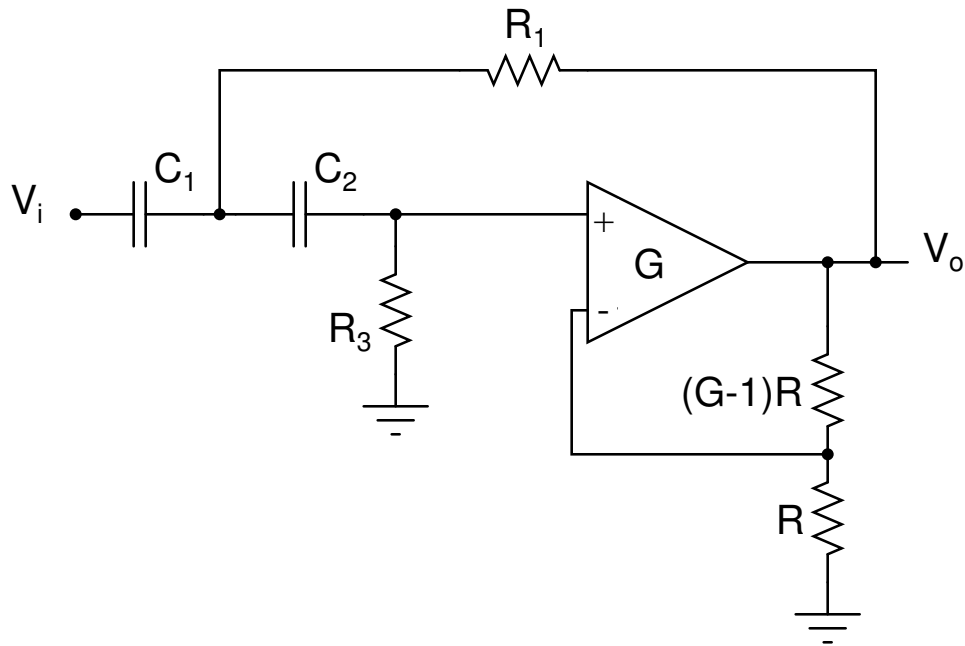
4 The Assignment

1. Obtain the step response of the circuit above. Use the signal toolbox of last week's assignment.
2. The input is

$$v_i(t) = \left(\sin(2000\pi t) + \cos(2 \times 10^6 \pi t) \right) u_0(t) \text{ Volts}$$

Determine the output voltage $v_0(t)$

Consider the following circuit (from page 274 of Horowitz and Hill)



The values you can use are $R_1 = R_3 = 10k\Omega$, $C_1 = C_2 = 1nF$, and $G = 1.586$.

3. Analyse the circuit using Sympy as explained above, i.e., create a function similar to the one defined above. For the same choice of components and gain, this circuit is a highpass filter.
4. Obtain the response of the circuit to a damped sinusoid (i.e., use suitable V_i). Use `signal.lsim` to simulate the output.
5. Obtain the response of the circuit to a unit step function. Do you understand the response? (Just define V_i to be $1/s$)